НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
В. о. завідувача кафедри
_____ О.Л. Тимощук

«___»_____2020 р

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»

на тему: «Аналіз ризиків в задачах інформаційної безпеки на основі апарата штучних нейронних мереж**»**

Виконав:
студент IV курсу, групи КА-64
Чумак Роман Євгенович _____
Керівник:
професор кафедри ММСА, д.т.н.
Мухін Вадим Євгенович _____
Консультант з економічного розділу:
к.е.н., доцент кафедри ТТПЕ
Шевчук Олена Анатоліївна _____
Консультант з нормоконтролю:
к.т.н., доцент кафедри ММСА
Коваленко Анатолій Єпіфанович _____
Рецензент:
доцент, к.т.н.,
доцент кафедри технічної кібернетики,
Корнага Ярослав Ігорович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 "Системний аналіз"

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Чумак Роман Євгенович

1. Тема роботи «Аналіз ризиків в задачах інформаційної безпеки на основі апарата штучних нейронних мереж», керівник роботи Мухін Вадим Євгенійович, професор, д.т.н. затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Дата видачі завдання _____

РЕФЕРАТ

Дипломна робота: 62 с. 16 рис., 11 табл., 1 додаток, 26 джерел

НЕЙРОННІ МЕРЕЖІ, АНАЛІЗ РИЗИКІВ, ІНФОРМАЦІЙНА БЕЗПЕКА, ОЦІНКА, ГЛИБОКЕ НАВЧАННЯ

Об’єкт дослідження - дані з National Vulnerability Database з інформацією про властивості відомих вразливостей комп’ютерних систем та Exploit-Database з інформацією про використання вразливостей на практиці.

Предмет дослідження - проблема класифікації та оцінки вразливостей комп’ютерних систем.

Мета дослідження – вирішення задачі бінарної класифікації вразливостей на «безпечні» та «небезпечні» за допомогою апарату штучних нейронних мереж глибокого навчання.

Методи дослідження - аналіз існуючих безкоштовних програмних методів оцінки захищеності системи та аналіз відкритих даних за допомогою нейронних мереж з оцінкою результатів різними метриками точності прогнозу.

Актуальність - надання можливості точної класифікації загроз за їх ознаками, що сприятиме ймовірному покращенню забезпеченності інформаційної безпеки з меншими затратами людських ресурсів на обробку даних задля експертної оцінки загрози.

Було проведено порівняльний аналіз з різними параметрами побудованої моделі.

Шляхи подальшого розвитку предмету дослідження - інтеграція програмного продукту з існуючими сканерами безпеки та більш широким діапазоном баз даних у відкритому доступі.

ABSTRACT

Thesis: 62 p. 16 fig., 11 tabl., 1 add., 26 sources

NEURAL NETWORKS, RISK ANALYSIS, INFORMATION SECURITY, EVALUATION, DEEP LEARNING

The object of the study is data from the National Vulnerability Database with information about the properties of known vulnerabilities in computer systems and Exploit-Database with information about the use of vulnerabilities in practice.

The subject of the research is the problem of classification and assessment of vulnerabilities of computer systems.

The purpose of the study is to solve the problem of binary classification of vulnerabilities into "safe" and "dangerous" using the device of artificial neural networks of deep learning.

Research methods - analysis of existing free software methods for assessing the security of the system and analysis of open data using neural networks with evaluation of results by different metrics of forecast accuracy.

Relevance - providing the ability to accurately classify threats by their characteristics, which will contribute to the likely improvement of information security with less human resources for data processing for expert assessment of the threat.

A comparative analysis with different parameters of the constructed model was performed.

Ways to further develop the subject - the integration of the software product with existing security scanners and a wider range of databases in the public domain.

ЗМІСТ

ВСТУП

Оскільки комп'ютерні системи стають все складнішими, необхідні значно вдосконалені методи для їх безпеки. Це стає особливо очевидним на рівні підприємства, де системні адміністратори стикаються з важкою проблемою забезпечення мережі з сотень чи тисяч підключених пристроїв. Якщо одна з цих систем була порушена, це може призвести до катастрофи для всього підприємства.

Одним з найбільш очевидних кроків, які може вжити адміністратор для захисту своєї мережі, є оновлення своїх систем останніми версіями від постачальників програмного забезпечення. Однак це набагато складніше завдання, ніж це здається. Велике підприємство може складатися з сотень чи тисяч робочих станцій, мобільних пристроїв, серверів, пристроїв зберігання даних, високопродуктивних комп'ютерів та іншої інфраструктури.

Через ці різні типи пристроїв працюють десятки чи сотні різних служб: послуги передачі файлів, поштові сервери, веб-сервери, бази даних, засоби віддаленого доступу, польові пристрої тощо. Кожна з цих служб може містити одну або кілька вразливих місць, які можуть бути використані для заподіяння руйнівної шкоди для системи та, можливо, всієї організації.

Хоча забезпечення роботи цих систем є доволі рутинним завданням, цим не можна нехтувати. Очевидним прикладом наслідків такого нехтування є атака програм WannaCry 2017 року. Протягом лише декількох днів це зловмисне програмне забезпечення поширилося на понад 230 000 комп'ютерів у 150 країнах [3]. Ця атака завдала незліченної шкоди, торкнувшись декількох секторів: енергетики, транспорту та логістики, зв'язку, судноплавства та, особливо важливо, охорони здоров'я.

Наразі існує величезна кількість різноманітних сканерів безпеки комп’ютерних систем та веб-додатків, які будуть розглянуті у першому розділі роботи. Використовуючи різні евристичні алгоритми, такі додатки доволі успішно знаходять вразливості систем. Основний недолік таких програм - це

те що аналіз результату їх роботи все одно потребує великих трудовитрат спеціалістів технічної безпеки для визначення коректності спрацювання і оцінки тієї чи іншої загрози. Саме тому, є актуальним введення допоміжного інструменту інтеллектуального аналізу оцінки безпеки системи у вигляді апарату нейронних мереж.

Метою даної дипломної роботи є створення програмного продукту, в основному модулі якого відбувається використання методик машинного навчання для створення інструменту, який аналізує відомі вразливості та класифікує їх відповідно до їх експлуатаційної можливості. Для забезпечення максимально можливої точності буде побудовано декілька різних моделей глибокої нейронної мережі.

РОЗДІЛ 1 ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1 Розгляд існуючих загроз інформаційній безпеці та їх типи

Інформаційна безпека системи - це така характеристика системи, яка характеризує, з одного боку, можливість протидіяти дестабілізуючому впливу зовнішніх та внутрішніх інформаційних загроз, а з іншої - рівень загроз, які створює її функціонування для елементів самої системи та зовнішньої середи [1].

Процес стрімкого розвитку інформаційних технологій завжди супроводжувався розвитком програмних забезпечень та різних видів кібератак, що становлять загрозу для комп’ютерних систем. Згідно з визначенням поняття інформаційної безпеки, загрозу для системи може становити також недосконаліть архітектури системи та помилки її проектування.

Класифікації загроз важливі, оскільки вони в основному дозволяють ідентифікувати та зрозуміти характеристики загроз та спосіб для захисту елементів системи. Більше того, за допомогою розуміння класифікації загрози, можна сформулювати ризики безпеки, які загрожують цим системам та обрати ефективне рішення для захисту системи.[2]

Тому, перш ніж розглядати існуючі варіанти вирішення проблеми аналізу ризиків інформаційної безпеки, спершу розглянемо безпосередньо класифікацію та деякі приклади загроз комп’ютерних систем.

Загроза - це мета зловмисника або те, що зловмисник може спробувати зробити системі. Загрозу можна також описати, як здатність зловмисника атакувати систему [6]. Таким чином, можна виділити два основні підходи до класифікаціїї загроз:

- Методи класифікації, що базуються на техніках кібератак
- Методи класифікації, засновані на впливі загроз.

В результаті визначення даних підходів, наведемо наступний список класифікаційних критеріїв загроз [2]:

- За джерелом загрози: походження загрози носить зовнішній або внутрішній характер;
- За фактором загрози: фактори загрози можна виділити у 3 основні класи - людські, екологічні та технологічні;
- За мотивацією загрози: мета зловмисників для системи може бути як шкідлива, так і не шкідлива;
- За наміром загрози: намір загрози може носити випадковий або навмисний характер;
- За наслідками впливу: можна визначити наступні наслідки загрози - знищення інформації, зміна інформації, крадіжка / втрата інформації, розголошення інформації, відмова роботи системи, зміна прав доступів та незаконне використання системи.

В даній роботі буде розглядатись та оцінюватись саме «вразливість» системи, тому доцільно буде розглянути деякі приклади зовнішніх та навмисних можливих кібератак. Можна виділити наступні типи останніх:

1. Перехват інформації. Даний тип атаки виникає тоді, коли зловмисник має можливість «проникати» у засіб передачі інформації і, таким чином, повідомлення надіслане з джерела А до приймача В доходить до зловмисника раніше ніж воно дійде до кінцевого пункту призначення. За наслідками впливу такий тип атаки може відноситись до зміни, крадіжки, втрати, розголошення інформації;
2. Атаки «грубою силою». До таких атак відноситься взлам систем шифрування методом підбору та інші подібні атаки.
3. DDoS (Distributed Denial of Service) атаки. Це тип атаки, який ставить під загрозу доступність даних тому що приводить систему до відмови в обслуговуванні. Виконується шляхом великої кількості відправлень команд до системи, які вона не в змозі обробити і стає непрацездатною;

4. «Шкідливе» програмне забезпечення. Це загальний термін, що описує типи шкідливого програмного забезпечення, який використовується зловмисником для порушення конфіденційності, доступності та цілісності даних. Найпоширеніші типи зловмисного програмного забезпечення: віруси, черв'яки, трояни, шпигунське програмне забезпечення;

5. Фішинг - це техніка, яка має на меті викрасти приватну інформацію у користувачів, маскуючись як надійне джерело (наприклад, веб-сайт) [7].

6. Впровадження операторів SQL— один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду [8].

1.2 Проблеми та недоліки традиційних засобів захисту інформації

Розвиток сучасних розподілених та різнорідних мереж, зростання складності та динамічності гетерогенного програмного забезпечення пов'язане з труднощами в розробці та застосуванні традиційних підсистем інформаційної безпеки. Однією з головних проблем таких підсистем є те, що вони орієнтовані на заздалегідь визначені класи загроз інформаційній безпеці [4]. Наведемо приклади проблемних питань, пов'язаних з деякими підсистемами інформаційної безпеки.

1. Ідентифікація та аутентифікація. Одним з прикладів дослідження проблем підсистем ідентифікації є проблема перевірки того, наскільки є стійкими засоби протидії автоматичному підбору паролей, засобів перевірки типу CAPTCHA, візуальний вигляд та алфавіт яких может постійно змінюватись;

2. Антивірусний захист. Сучасні засоби захисту від вірусів, що використовують сигнатурні методи, на сьогодні перебувають в стані серйозної кризи. За оцінками деяких спеціалістів, сучасні антивірусні засоби не в змозі захистити комп’ютерні системи від 80% небажаних програм. Особливо наочно

дане твердження підтвердили факти зараження троянськими програмами, які антивірусні засоби не могли виявити протягом 5-8 років [5];

3. Виявлення та попередження вторгнень. Системи виявлення комп'ютерних атак, як і антивірусні засоби, головним чином, орієнтовані на сигнатурні («шаблонні») методи, які є неефективними перед новими класами атак;

4. Керування ризиками інформаційної безпеки. При проведенні аналізу ризиків, виникає проблема неможливості опису ряду факторів безпеки кількісною мірою. Тим більше, багато факторів мають зовсім різну природу та закономірності їх проявів.

Даний список може бути продовжений, але стає очевидною потреба у впровадженні нових методів забезпечення інформаційної безпеки.

1.3. Розгляд існуючих сучасних методів аналізу ризиків інформаційної безпеки та методи їх роботи

Наразі є доступними величезна кількість різноманітних систем моніторингу безпеки комп’ютерних мереж або програмного забезпечення. До таких систем відносять сканери вразливостей. До сканерів вразливостей можна віднести:

- сканери портів;
- сканери, що досліджують топологію комп’ютерної мережі;
- сканери, що досліджують вразливості мережевих сервісів;
- CGI-сканери (сканери, що можуть допомогти знайти вразливі частини коду).

Сканер відкритих портів - програмний засіб, створений для пошуку хостів мережі, в яких відкриті потрібні порти. Сканування портів може бути першим кроком у процесі упередження взлому системи, тому що це може допомогти виявити доступні цілі кібератаки. До найбільш відомих, на

сьогодняшній день, сканерів портів можна віднести Advanced Port Scanner, Nmap, IPVOID та інші.

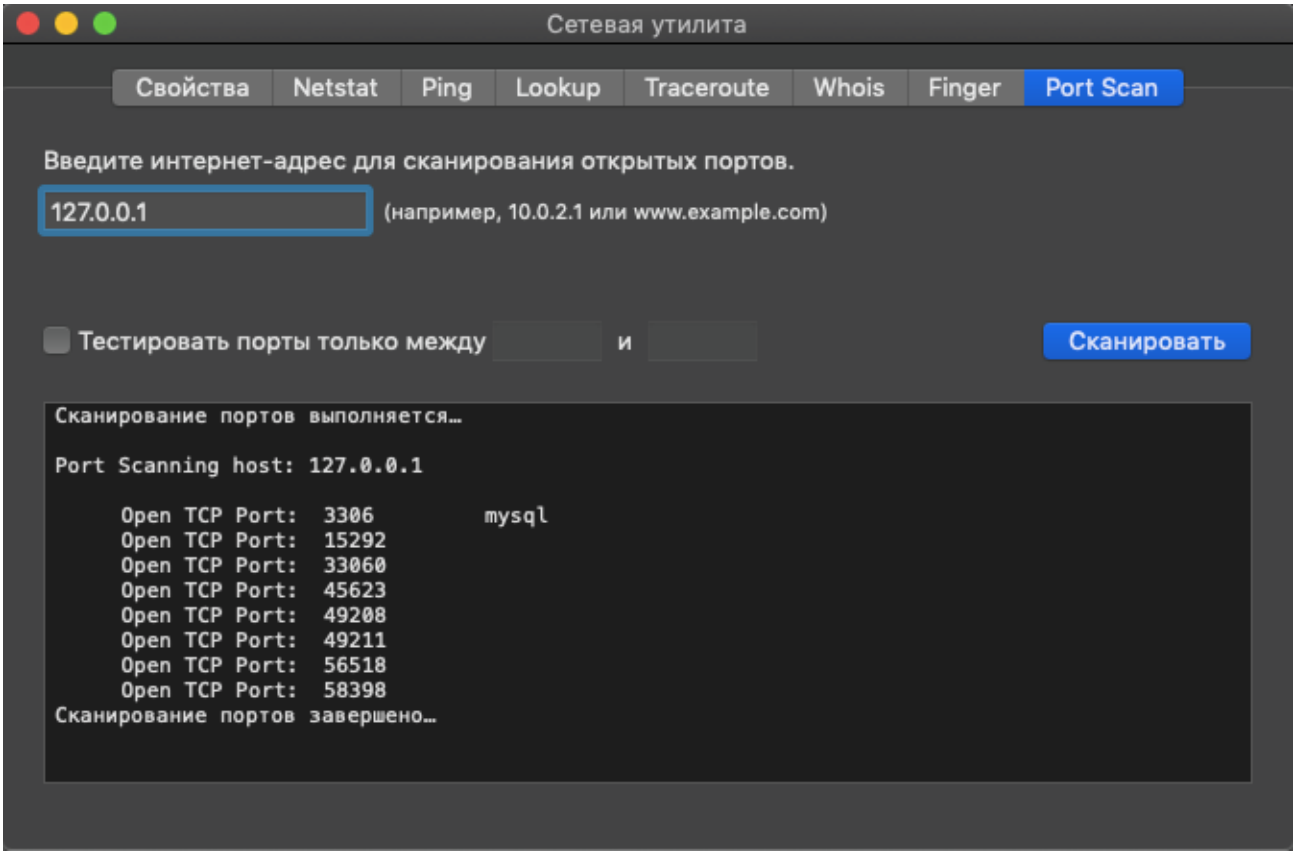

Рисунок 1.1 - Приклад використання додатку Network Utility

В операційній системі Windows для цього може використовуватися також командна строка - cmd.exe, що є вбудована в систему. Приведемо приклад роботи аналогічного інструменту на платформі Mac OS - Network Utility. На рис. 1.1 можемо бачити результати сканування локального хосту (localhost). Бачимо, що дана утіліта дає можливість також обрати діапазон портів для пошуку, що є доволі зручно та економить час при пошуку. В результаті роботи Network Utility, можемо побачити перелік відкритих TCP портів на локальному хості.

Щодо сканерів вразливостей веб-сервісів та сканерів інфраструктури - до найбільш відомих та ефективних продуктів відносять: Nessus, X-Spider,

OpenVAS, MaxPatrol, OWASP ZAP, Internet Scanner, IBM AppScan та багато інших.

В різних сканерах використовуються алгоритми евристичного сканування, тобто аналіз послідовності команд в об’єкті, що перевіряється, набір деякої статистики та прийняття рішень («можливо вражений», «не вражений») для кожного перевіряємого об’кту [14]. Оскільки евристичне сканування багато в чому є ймовірністним методом пошуку вразливостей, то на нього розповсюджуються закони теорії ймовірностей. Наприклад, чим вищий процент знаходження вразливостей - тим більший процент «хибних спрацьовувань». Тобто, такі методи забезпечують 100-відсоткового коректного виявлення загроз.

Також, такі сканери вразливостей, в основному, використовують методології тестування BlackBox testing або WhiteBox testing. Іноді використовується метод GreyBox testing, що поєднує у собі особливості обох попередніх.

Перевірка "чорного ящику" - це така методологія тестування програмного забезпечення, при якому функціональність досліджується без розглядання коду, деталей реалізації та знань про те, як програмне забезпечення (ПЗ) влаштоване всередині. Метод імітує поведінку користувача який не має знань про внутрішню структуру. [13] При blackbox-скануванні - обраний інструмент повинен мати можливість працювати з сервісом чи мережею через ті самі інтерфейси, через які з ним працюють користувачі.

Сканери інфраструктури (Tenable Nessus, Qualys, MaxPatrol, Rapid7 Nexpose и т.д.) [12] шукають відкриті мережеві порти, збирають «банери», визначають версії встановленого ПЗ і шукають у своїй базі знань інформацію про вразливості в цих версіях. Також, вони намагаються знайти помилки конфігурації, такі як паролі за замовчуванням або відкритий доступ до данних, слабкі шифри SSL та інше.

Сканери веб-додатків (Acunetix WVS, Netsparker, Burp Suite, OWASP ZAP, і т.д.) [12] також вміють визначати відомі компоненти та їх версії

(наприклад, фреймворки, JS - бібліотеки). Основні кроки сканера - це краулінг та фазер.

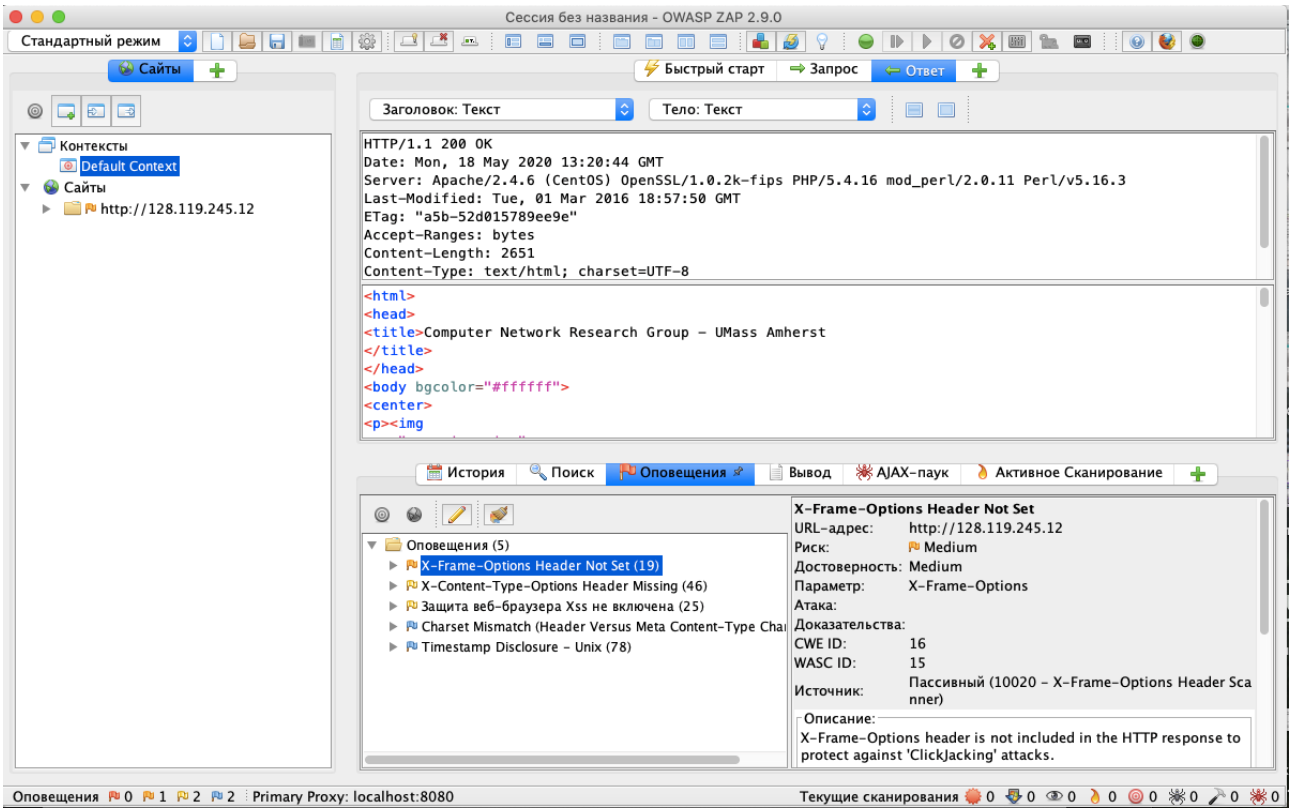

Рисунок 1.2 - Вигляд сканеру OWASP ZAP у процесі роботи

В ході краулінгу сканер збирає інформацію про всі існуючі інтерфейси додатку, HTTP-параметрах. В ході фазингу - у всі знайдені параметри підставляються змінені або сгенеровані дані, що мають своєю ціллю спровокувати помилку та знайти вразливість. Такі сканери додатків відносять до класів DAST та IAST - відповідно Dynamic та Interactive Application Security Testing.

Щодо тестування білого ящику (WhiteBox testing), то у цього методу є декілька назв («скляний ящик», «відкритий ящик» та інші). Перевіка «білого ящика» - це метод тестування програмного забезпечення, який припускає, що внутрішня структура і реалізація системи відоми тестуючій стороні. В рамках процесу, VM сканерам (Vulners, Incsecurity Couch, Vuls, Tenable Nessus і т.д.) часто дають доступ до систем, проводячи аутентифікований скан. Таким

чином, сканер може вивантажити встановлені версії пакетів і конфігураційні параметри прямо з системи, не вгадуючи їх по банерам мережевих сервісів.

Скан виходить більш точний і повний. Якщо ж говорити про whitebox-сканування додатків (CheckMarx, HP Fortify, Coverity, RIPS, FindSecBugs і т.д.), то мова зазвичай йде про статичний аналіз коду і використанні відповідних інструментів класу SAST - Static Application Security Testing.

В рамках даної роботи, наведемо приклад роботи безкоштовного, але тим не менш, доволі ефективного, сканера веб-додатків OWASP ZAP (приклад роботи сканера зображений на рис. 1.2.). Основні можливості даного додатку:

- Man-in-the-middle Proxy - пошук атак посередника;
- Звичайний та AJAX Spider. Spider - це інструмент пошуку прихованих URL на обраному для аналізу сайті;
- Автоматичний сканер;
- Пасивний сканер;
- Forced browsing - даний термін описує ситуацію, коли користувач може отримувати доступ до контенту сайту, який повинен бути недоступний;
- Фазер.

Інтерфейс програми складається з вікна «дерева» проаналізованих веб-сайтів, робочого простору та вікна виводу інформації.

Був протестований хост за IP-адресою: 128.119.245.12. Можемо бачити, що сканер у стандартному режимі виявив 5 вразливостей: одній він присвоїв Середній ступінь ризику (Medium), а двом іншим - Низький ступінь ризику (Low). Також, є дві вразливості що носять інформаційний характер (Informational). У вкладці «Активне сканування», ми можемо знайти інформацію про відправлені та отримані сканером пакети на обрану адресу.

1.4 Огляд деяких оцінок степені ризику

Питання класифікації степені ризику, пов’язаного з вразливістю додатків або мереж є важливою темою. Зазвичай, оцінка ризикованих елементів комп’ютерної системи і є результатом роботи подібних сканерів вразливості. В більшості своїй, різні сканери мають власні метрики оцінювання небезпек, а не використовують якусь одну загальноприйняту [9]. Найбільш розповсюджені наступні підходи до аналізу степеня ризику:

- класична оцінка, що виділяє вразливості «високої», «середньої» та «низького» степеня ризику;
- п’ятирівнева модель, що прийнята у стандарті PSI DSS [10] та визначає рівні наступним чином: «критичний», «невідкладний», «високий», «середній» та «низький» (Urgent, Critical, High, Medium, Low);
- метод Common Vulnerability Scoring System (CVSS) [11], що оцінює степінь ризику як число від 0 до 10. При використанні у сканерах вразливостей, дане число трансформується також у своєрідну п’ятирівневу модель (CVSS v3.0) [11]:

Таблиця 1.1 - Рейтингова таблиця оцінки CVSS

Загроза	**Оцінка**
Немає	0.0
Низька	0.1 - 3.9
Середня	4.0 - 6.9
Висока	7.0 - 8.9
Критична	9.0 - 10.0

1.5 Висновок до розділу 1

В даному розділі надана класифікація загроз для комп’ютерних систем, а також наведені найбільш розповсюджені різновиди зовнішніх кібератак. Також, були розглянуті розповсюджені метрики, що використовуються для оцінки захищеності мереж та веб-додатків.

В рамках даного розділу, був проведений огляд найбільш розповсюджених, на сьогодняшній день, інструментів, що використовуються для тестування ступеня захищеності та оцінки вразливості комп’ютерних систем. Також, були описані методології, що використовуються у цих інструментах.

Основним результатом, що очікується від роботи таких сканерів є список «вразливих» кандидатів, елементів системи, що аналізується. Головним недоліком інструментів такого класу - це використання складних евристичних алгоритмів, що приводить до великої кількості числа хибних спрацювань (false positives) та заповненню списку тими вразливими елементами, які, насправді, не існують у реальному об’єкті аналізу. В зв’язку з цим - після роботи таких сканерів потребується серйозна та трудомістка робота експертів-аналітиків з інформаційної безпеки, щоб перепровірити, підтвердити або спростувати кандидатів у вразливості [15].

Таким чином, залишається відкритим питання валідації вразливостей, що знайдені за допомогою сканерів. В цьому може бути корисним саме апарат штучних нейронних мереж.

РОЗДІЛ 2 ТЕОРЕТИЧНА ОСНОВА ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ ТА ОПИС МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ КЛАСИФІКАЦІЇ ВРАЗЛИВОСТЕЙ

2.1 Штучні нейронні мережі

Під штучними нейронними мережами (НМ) розуміють сукупність моделей біологічних нейронних мереж. Структурно НМ є мережею елементів - штучних нейронів, пов'язаних між собою синоптичними сполуками. НМ обробляє вхідну інформацію і в процесі зміни свого стану в часі формує сукупність вихідних сигналів.

Кожен нейрон характеризується своїм поточним станом за аналогією з нервовими клітинами головного мозку, які можуть бути порушені або загальмовані. Він володіє групою синапсів - однонаправленних вхідних зв'язків, з'єднаних з виходами інших нейронів, а також має аксон вихідний зв'язок даного нейрона, з якої сигнал (збудження або гальмування) надходить на синапси наступних нейронів. Узагальнена математична модель нейрона виглядає наступним чином (рис. 2.1) [15]:

У даній моделі наявні:

- вхідні сигнали x_i - дані, що надходять з навколишнього середовища або від інших активних нейронів. Рівні введення можуть бути дискретними з множин [0, 1] або [-1, 1] або приймати будь-які дійсні значення;
- дійсні вагові коефіцієнти w_i - визначають силу зв'язку між нейронами;
- рівень активації (потенціал) нейрона (2.1);
- функція активації $f(P)$ - призначена для обчислення вихідного значення сигналу, що передається іншим нейронам.

$$P = \sum_{i=1}^{n} w_i x_i \qquad (2.1)$$

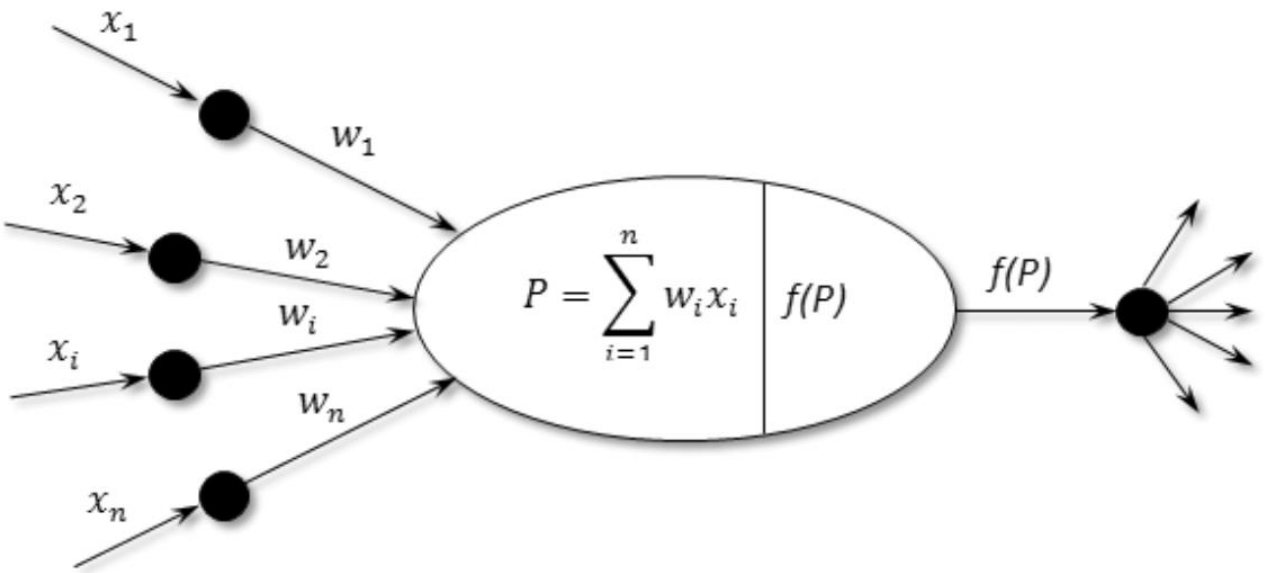

Рисунок 2.1 - Узагальнена модель нейрона

Штучна нейронна мережа складається з декількох шарів, в які групуються нейрони. Схематично її можна представити таким чином: (рисунок 2.2) [15].

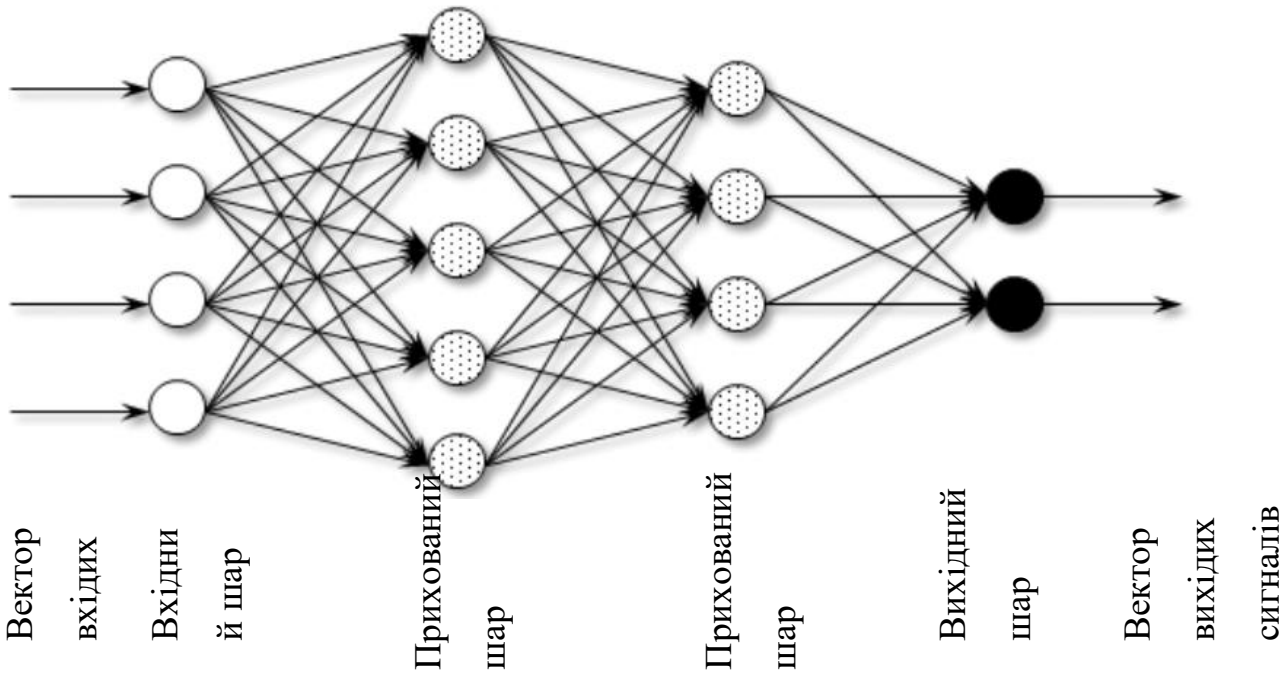

Рисунок 2.2 - Штучна нейронна мережа

На нейрони вхідного шару подаються виміряні і перетворені значення характеристик об'єкта (образу). Сигнали, що надійшли до нейронам вхідного шару, передаються на наступний шар (прихований або вихідний) без перетворення (тобто до них не застосовується функція активації). Приховані шари і вихідний шар, що відображають специфіку знань, перетворять вхідні дані. Вихідний шар генерує скаляр (одне значення) або вектор (кілька значень), що є вирішенням поставленної задачі. Вибір структури НС здійснюється відповідно до особливостей і складності завдання.

Залежно від кількості шарів, що виконують перетворення (прихованих і вихідного), розрізняють одно- (в мережі відсутні приховані шари) і багатошарові мережі.

Залежно від функції активації розрізняють наступні ШНМ (Табл. 2.1), де:

- P^*, P_1^*, P_2^* - порогові значення;
- a, b - коефіцієнти;
- e - основа натурального логарифму.

Таблиця. 2.1 - Типи ШНМ залежно від функції активації

Функція	**Спосіб визначення Y**
Ступінчаста порогова	$Y = 0$ при $P < P^*$ $Y = 1$ при $P \geq P^*$
Лінійна порогова	$Y = 0$ при $P < P_1^*$ $Y = a + bP$ при $P_1^* \leq P^* \leq P_2^*$ $Y = 1$ при $P \geq P_2^*$ $P_2^* = P_1^* + 1/b$
Сигмоїдальна	$Y = 1/(1 + e^{-a(P-P^*)})$
Гіперболічний тангенс	$Y = a * th(bP)$ $= a * (e^{bP} - e^{-bP})/(e^{bP} + e^{-bP})$

Функція	Спосіб визначення Y
Арктангенс	$Y = 2 * arctg(P)/\pi$
Лінійна	$Y = a + bP$
Гаусіан	$Y = e^{-b(P-P^*)}$

Залежно від типу міжнейронних зв'язків розрізняють ШНМ:

- з прямими зв'язками;
- з перехресними зв’язками;
- із зворотними (рекурентними) зв'язками. У таких мережах нейрон може посилати сигнали сам собі, нейронам того ж шару або нейронам попередніх шарів.

Перед безпосереднім використанням, тобто перед рішенням конкретного завданн, необхідно налаштувати (навчити) мережу. Навчання НМ - це процес, в якому вільні параметри нейронної мережі настроюються за допомогою моделювання середовища, в яку ця мережа вбудована. Тип навчання визначається способом підстроювання цих параметрів. Процес навчання мережі полягає у визначенні набору зв'язків і коефіцієнтів зв'язків між нейронами.

Існує безліч різних алгоритмів навчання, які діляться на два великі класи:

детерміновані та стохастичні. У першому з них підстроювання ваг являє собою жорстку послідовність дій, у другому - вона проводиться на основі дій, що підкоряються деякому випадкового процесу.

Залежно від способу навчання розрізняють такі типи ШНМ:

- ті, яких навчають з учителем (контрольоване навчання). При навчанні з учителем всеі приклади навчальної вибірки містять правильні відповіді (виходи), що відповідають вихідним даним (входів). В процесі навчання ваги (коефіцієнти) налаштовуються так, щоб мережа породжувала відповіді, найбільш близькі до правильних;

- ті, яких навчають без вчителя (неконтрольоване навчання). Навчання без учителя використовується, коли не для всіх прикладів навчальної вибірки відомі правильні відповіді. В цьому випадку робляться спроби визначення коефіцієнтів мережі з метою визначення категорій (класів) зразків і подальшого їх розподілу за категоріями. Використовується, зокрема, для вирішення завдань кластеризації;

- зі змішаним (гібридне) навчанням. При змішаному навчанні частина ваг визначається за допомогою навчання з учителем, а інша частина виходить за допомогою алгоритмів самонавчання.

Процедура побудови штучних нейронних мереж, які використовуються в задачах класифікації, включає наступні етапи.

Етап 1. Визначення типу образів, що розпізнаються (фотографія, відео, сукупність числових характеристик і т.п.), набору вимірюваних параметрів образів, інформативних з точки зору розпізнавання, і класів розпізнаваних образів.

Етап 2. Вибір способу перетворення (застосування фільтрів, зменшення шумів, сегментація і т.п.) вимірюваних параметрів образів у вхідний вектор чилових величин і представлення класів образів у вигляді вектора вихідних величин. Ефективність нейросетевой моделі підвищується, якщо діапазони зміни величин вхідного і вихідного векторів будуть нормалізовані, наприклад, в діапазоні [0,1] або [-1,1].

Етап 3. Проектування архітектури штучної нейронної мережі:

- визначення кількості шарів. У мережі повинно бути, як мінімум, два шари: вхідний і вихідний. Кількість прихованих шарів визначається, як правило, експертним або шляхом досвіду;

- визначення кількості нейронів в кожному шарі. Кількість нейронів у вхідному шарі відповідає кількості перетворених вимірюваних параметрів образів, що розпізнаються. Кількість нейронів у вихідному шарі, як правило, відповідає кількості класів розпізнаваних образів. Визначення

кількості нейронів в кожному прихованому шарі є неформальною проблемою, при вирішенні якої можна використовувати евристичне правило: число нейронів в наступному шарі повинно бути в два рази менше, ніж в попередньому;

- вибір типу зв'язків між нейронами. Залежить від специфіки розв'язуваної задачі;
- вибір функції активації. Залежить від специфіки розв'язуваної задачі.

Етап 4. Навчання мережі - уточнення значень вагових коефіцієнтів зв'язків на основі багаторазового прогону навчальних прикладів через мережу. Кожному навчальному прикладу (образу) строго відповідають певні вектора вхідних і вихідних величин. Навчальні приклади, як правило, представляють собою еталонні (ідеальні) уявлення образів, а також їх незначні модифікації. Вони повинні бути підібрані для кожного класу образів. Існує емпіричне правило, яке встановлює рекомендований співвідношення λ між кількістю навчальних прикладів і числом зв'язків в нейронної мережі: $\lambda \leq 10$ [15].

Етап 5. Тестування мережі за допомогою контрольного набору прикладів (образів) для оцінки якості обраної архітектури та ступеня навчання. Контрольний набір повинен містити, як мінімум, по одному прикладу для кожного класу образів і включати в себе як еталонні, так і значно спотворені образи.

Етапи з третього по п'ятий можуть спільно застосовуватися для уточнення архітектури мережі на основі конструктивного або деструктивного підходу. Відповідно до першого підходу навчання починається на мережі невеликого розміру, який поступово збільшується до досягнення необхідної точності за результатами тестування. Деструктивний підхід базується на принципі «віджигання дерева», відповідно до якого з мережі з явно надмірною об'ємом поступово видаляють «зайві» шари, нейрони і примикають до них зв'язку. Цей підхід дає можливість досліджувати вплив видаляються елементів на точність розпізнавання мережі.

2.2 Опис використаних даних

Одним з найважливіших елементів роботи з нейронними мережами є процес вибору та препроцесінгу вичерпного набору навчальних даних із безлічі джерел відкритого доступу. Модель машинного навчання буде настільки ж хороша, як і дані, які використовуються для її навчання; тому складання корисного набору даних є критично важливим. Для створення навчальних даних використовуються декілька онлайн-баз даних, які були обрані завдяки релевантності та доступності їх даних. Обидві бази даних надають свої дані у форматі XML або JSON, що особливо спрощує процедуру їх підготовки для розбору в єдиний набір навчальних даних. Вибрані два джерела - National Vulnerability Database (NVD) та Public Exploit Database.

National Vulnerability Database (NVD), опублікована Лабораторією інформаційних технологій Національного інституту стандартів і технологій США (NIST), є офіційним сховищем урядових американських сховищ даних управління вразливістю на основі стандартів. Дані представлені за допомогою протоколу автоматизації контенту безпеки, який дозволяє використовувати дані для автоматизації управління вразливістю, вимірювання безпеки та відповідності. Він містить велику кількість даних про десятки тисяч дискретних вразливостей, включаючи оцінки впливу, вектори атак, класифікації, програмне забезпечення та інше. Поля, вибрані як функції для набору даних, перелічені нижче:

- Access Vector: напрямок, через який вразливість дозволяє отримати доступ до системи. Можливі значення: Network, Adjacent, Local, Physical (Мережа, Суміжні, Локальні, Фізичні);
- Access complexity: складність доступу, що створена вразливістю. Можливі значення: Low, High (Низький, Високий);

- Confidentiality impact: потенційний вплив вразливості на конфіденційність системи. Можливі значення: None, Low, High (Жодне, Низьке, Високе);

- Integrity impact: потенційний вплив вразливості на цілісність системи. Можливі значення: None, Low, High (Жодне, Низьке, Високе);

- Availability impact: потенційний вплив вразливості на доступність системи. Можливі значення: None, Low, High (Жодне, Низьке, Високе);

- CVSS score: оцінка за шкалою Common Vulnerability Scoring System (CVSS), загальна основа для ранжування потенційного впливу вразливих місць. Можливі значення: 0,0-10,0. Інформація про дану оцінку була надана у першому розділі;

- Impact score: оцінка за шкалою від 0 до 10. Показник, що подається також на виході розрахунку оцінки CVSS та відображує можливий вплив вразливості на комп'ютерне середовище;

- Exploitability score: оцінка за шкалою від 0 до 10. Показник, що такж подається на виході розрахунку оцінки CVSS та відображує можливість використання загрози.

Ці поля були обрані насамперед для зручності кодування їх як числових значень, які можна використовувати як функції для моделі (процес кодування буде описаний у розділі 3). Можливість додавання додаткових функцій до бази даних буде описане у висновку до роботи. Як уже згадувалося раніше, усі записи в базі даних доступні для завантаження у форматі XML або JSON, що робить його ідеальним для локального дзеркального відображення та обробки у форматі, придатному для навчання моделі машинного навчання.

Друга база даних, яка використовується при складанні навчальних даних, - це Exploit Database (Exploit-DB), опублікована Offast Security, навчальною компанією з інформаційної безпеки, яка надає сертифікати інформаційної безпеки, а також послуги з тестування проникнення. На відміну від NVD, Exploit-DB не надає функцій для використання навчальної моделі -

він постачає мітки, що використовуються для формування вихідного вектору набору навчальних даних, придатних для навчання «з учителем». Без міток, наданих Exploit-DB, підхід навчання «з учителем» був би неможливим. NVD надає інформацію про вразливість, але ця інформація не є корисною, якщо вона не може бути пов'язана з наявністю (або відсутністю) існування експлуатації в будь-яких цілях тієї чи іншої вразливості. Забезпечивши метод перехресного посилання на вразливість від NVD із відомим її використанням з Exploit-DB, може бути створений навчальний набір, який містить для кожної вразливості інформацію про цю вразливість та про те, використовувалась ця вразливісті чи ні «на практиці». Це забезпечує повний набір мічених даних про навчання, які потім можуть бути використані у підході навчання «з учителем».

Метод співвідношення двох баз даних фактично надається тією ж організацією, яка забезпечує NVD. NIST підтримує основу для ідентифікації вразливостей, відомих як список Common Vulnerabilities and Exposures (CVE). Цей фреймворк використовується для забезпечення кожної дискретної вразливості з ідентифікаційним номером, починаючи з "CVE-XXXX-YYYY", де XXXX - рік, коли було зафіксовано вразливість, а YYYY - номер конкретної вразливості [16]. CVE також надає набір довідкових карт, які співвідносять ідентифікаційні номери CVE з ідентифікаційними номерами інших відомих служб. Одна з референтних карт призначена для Exploit-DB, яка забезпечує основу для зіставлення відомих вразливостей з відомими їх використаннями для створення навчальної вибірки. На жаль, довідкова карта недоступна у форматі XML або JSON, тому на мові Python був написаний парсер даних з веб-сторінки (використовуючи бібліотеки Requests та BeautifulSoup), щоб спарсити та проаналізувати довідкову карту у зручному форматі. Після цього було використано інший скрипт для ітерації вмісту каналу даних NVD, та перевірки наявності певної загрози у базі Exploit-DB.

2.3 Опис нейронної мережі глибокого навчання

Deep Neural Network (DNN) - це спеціалізована форма штучної нейронної мережі (ШНН), типу обчислювальної системи, призначеної для грубої емуляції біологічних нейронних мереж, наявних у мозку тварин [17].

DNN добре підходять для проблем класифікації, завдяки їх здатності класифікувати нелінійні вхідні дані [18]. Однак DNN не є ідеальними: до потенційних недоліків можна віднести перевищення рівня навчальних даних та часу на обчислення. Перенавчання - це проблема, коли модель надто близько відповідає наданим навчальним даним та не спроектує сценарій реального світу (продуктивність значно знижується при впровадженні нових даних) [19]. Можуть бути вжиті заходи для запобігання перенавчанню, включаючи обмеження кількості прихованих шарів та маніпулювання навчальними даними, але це все ж залишається проблемою використання DNN. По-друге, підготовка DNN потребує значних витрат часу та ресурсів для обчислення, особливо при використанні сотень тисяч наборів даних (що рекомендується для більшої точності) [20]. Для скорочення цього часу можна використовувати декілька методів, наприклад, обчислення «пакетами» (обчислення градієнта на декількох наборів даних одночасно, а не окремо), але це все одно не є панацеєю, особливо в сценаріях, де доступні обмежені ресурси для обчислень.

2.4 Опис архітектури нейронної мережі

Після вибору набору даних наступним кроком у процесі проектування є архітектура структури нейронної мережі, що використовується для навчання та оцінки даних. Оскільки нейронні мережі є евристичними за своєю природою, немає єдиної прямої відповіді, яка архітектура буде найкраще працювати - це вимагає експериментування та вдосконалення. Однак є кілька

основних проектних рішень, які мають бути прийняті у процесі впровадження мережі.

Перше важливе рішення полягає в тому, чи повинна мережа бути з рекуррентними або з прямими зв’язками (інформація, що рухається лише від вводу до виходу). Рекуррентні мережі в основному використовуються тоді, коли мережі необхідно зберігати інформацію про те, про що вона дізналася в минулому, що особливо не відноситься до даної реалізації [20]. Крім того, використання мереж з прямими зв’язками, дозволяє використовувати метод зворотнього поширення помилки - процес, коли мережа обчислює помилку на вихідному шарі і поширює її назад по мережі з метою виправлення ваг шару [21]. Мережі з прямими зв’язками вважаються загальноприйнятними для даного типу проблем класифікації, тому ця конструкція використовується при реалізації мережі.

Друге рішення стосується кількості прихованих шарів. Всі нейронні мережі містять щонайменше два шари: один вхідний і один вихідний. Визначальною особливістю нейронної мережі глибокого навчання (тип мережі, обраний для даної роботи) є наявність додаткових шарів між входом і виходом, відомих як приховані шари. Кількість та структура цих шарів часто визначається шляхом експерименту, а не проектування, але переважна більшість проблем вирішуються достатньо продуктивно з одним прихованим шаром, а додавання більшої кількості шарів лише збільшує ймовірність для перенавчання (помилка де модель занадто близько підходить до набору навчальних даних і не вдається точно моделювати реальні процеси) [22]. Тому, в процесі роботи, було прийнято рішення почати з одного прихованого шару та вдосконалювати модель в міру необхідності.

Після того, як ця основна структура буде визначена, для кожного шару потрібно вибрати точну кількість вузлів та функцію активації. Визначення даної кількості, знову ж таки, це число відкрите для експериментів, але є кілька загальноприйнятих вказівок щодо вибору правильної кількості вузлів. Як правило, розмір прихованого шару повинен бути між розмірами вхідного та

вихідного шарів (ці розміри визначаються набором даних, про який буде сказано докладніше наступному розділі). Крім того, кількість прихованих вузлів повинна приблизно дорівнювати 2/3 від розміру вхідного шару плюс розмір вихідного шару. Нарешті, кількість прихованих нейронів має бути меншою, ніж кількість нейронів вхідного шару, помножена на 2 [23]. Усі ці вказівки більше відносяться до рекомендацій, ніж до правил, але вони дають зручну вихідну точку для методу проб і помилок, який будує доволі точну нейронну мережу глибокого навчання. Для вхідного шару кількість вузлів еквівалентна кількості функцій у навчальних даних (у нашому випадку - 8). Для вихідного шару потрібен лише один вузол, для вирішення поставленої проблеми бінарної класифікації. Для прихованого шару приймається рішення використовувати 6 або 4 нейрони, які потім можна було б скоректувати експериментальним шляхом.

Нарешті, для кожного шару потрібно вибрати функцію активації. Це функція, яка обчислює ваги кожного вузла і ребра після кожної епохи тренувань. Це також є ще один аспект DNN, який можна регулювати. Для нелінійної задачі бінарної класифікації, яка вирішується в даній роботі, функція (ReLU) вважається відповідним вибором [26]. Дана функція визначається наступним чином:

$$f(x) = x^{+} = max(0, x)$$

де *x* - вхідне значення нейрона. Графік функції відображений на рисунку 2.3. Дана функція використовується у вхідному та прихованому шарі.

На вихідному шарі, в якості функції активації використовується сигмоїдальна функція (2.2) де *x* - вхідне значення нейрона.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Крім того, необхідно вибрати функцію втрат (функція обчислення втрат моделі після кожної епохи) та метрику точності (метрику для обчислення точності передбачення кожної епохи).

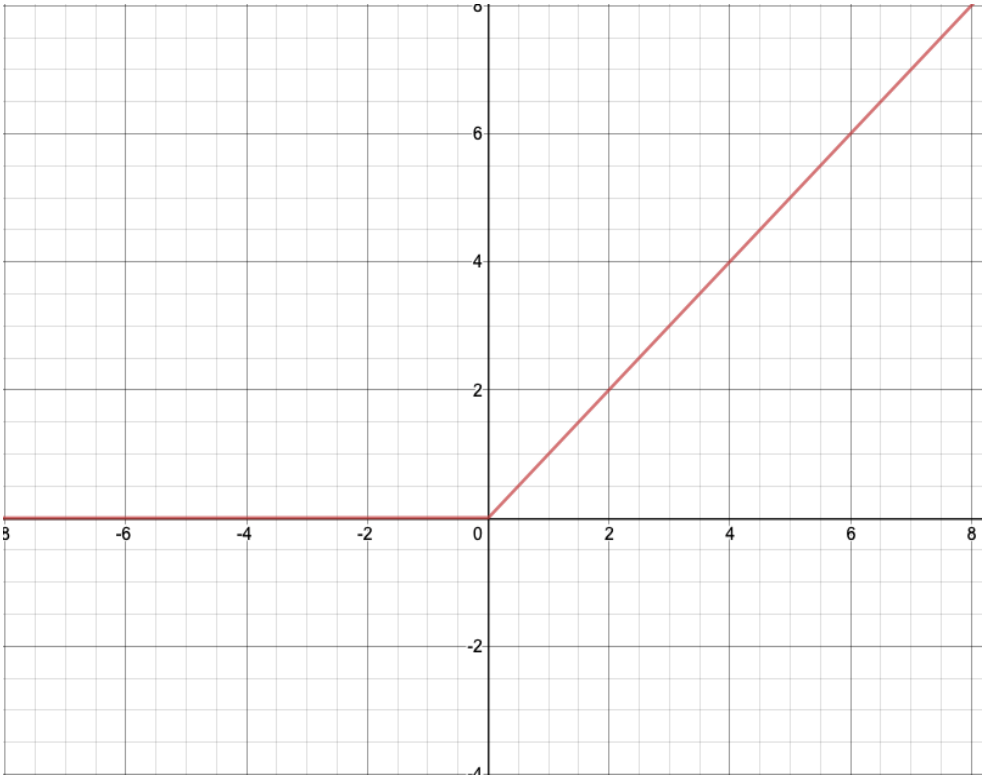

Рисунок 2.3 - Графік функції ReLU

Для проблеми бінарної класифікації керівництво користувача для фреймворку, що використовується при реалізації, пропонує функції "binary_crossentropy" та "binary_accuracy" (2.6) відповідно [24].

Функція втрат binary_crossentropy розраховується визначається у (2.3).

$$L\left(y,\hat{y}\right)=\frac{1}{n}\sum_{i=1}^{n}y_i*ln\left(1-\hat{y}_i\right)+(1-y_i)*ln\left(\hat{y}_i\right) \quad (2.3)$$

У розділі 3 на результатах роботи моделі буде показано, що однієї метрики точності є замало для оцінки адекватності моделі. Тому введемо, також, наступні метрики: precision (2.4) та recall (2.5).

Для того, щоб навести формули даних метрик, потрібно ввести поняття матриці помилок та її елементів (табл. 2.2).

Таблиця 2.2 - Опис матриці помилок

Прогноз	**Реальність**	
	True	**False**
True	True Positive (істинне позитивне рішення): співпадіння прогнозу з реальністю, стався позитивний результат як і було передбачено моделлю.	False Positive (хибно-позитивне рішення): помилка 1го роду, модель передбачила позитивний результат, а насправді він негативний
False	False Negative (хибно-негативне рішення): помилка другого роду - модель передбачила негативний результат, а насправді він позитивний	True Negative (істинно-негативне рішення): результат негативний, співпадіння прогнозу моделі з реальністю

Формули метрик:

$$precision = \frac{TP}{TP + FP} \tag{2.4}$$

$$recall = \frac{TP}{TP + FN} \tag{2.5}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

Міра precision (точність) характеризує, скільки отриманих від класифікатора позитивних відповідей є правильними. Чим більше точність, тим менше число помилкових влучень. Міра точності, однак, не дає уявлення про те, чи всі правильні відповіді повернув класифікатор. Для цього існує так звана міра повноти (recall). Міра повноти характеризує здатність класифікатора «вгадувати» якомога більше число позитивних відповідей з очікуваних.

2.5 Висновок до розділу 2

У даному розділі була детально розглянута теоретична основа апарату нейронних мереж. Надана классифікація різних типів нейронних мереж.

Також, у другому розділі даної роботи були описані відкриті ресурси даних, ознаки та поля що використовувались для формування кінцевого датасету, який подається на вхід нейронної мережі. Наведені метрики, що будуть використані для аналізу якості відповідей нейронної мережі глибокого навчання та означена архітектура, функції активації ШНН у першому наближенні.

Основним результатом, що очікується від теоретичної основи що закладена у цьому розділі - висока ефективність та точність моделі на експериментальних даних.

РОЗДІЛ 3 ОПИС ОБРОБКИ ВХІДНИХ ДАНИХ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ НЕЙРОННОЇ МЕРЕЖІ ГЛИБОКОГО НАВЧАННЯ

3.1 Обробка вхідних даних

Першим кроком у реалізації обраної моделей є збір даних із двох джерел та створення з них єдиного набору навчальних даних.

Таблиця 3.1 - Кодування вхідних даних у цілочисельні поля

Поле	**Строкове значення**	**Відповідне цілочислове значення**
Access Vector	Network	1
	Adjacent	2
	Local	3
	Physical	4
Access Complexity	Low	1
	High	2
Confidentiality Impact	None	1
	Low	2
	High	3
Integrity Impact	None	1
	Low	2
	High	3
Availability Impact	None	1
	Low	2
	High	3

Для NVD (National Vulnerability Databese) збір даних є досить простим: NVD надає канали даних кожного року у вигляді файлів JSON та XML, які можна завантажити, та які містять інформацію про всі виявлені вразливості за

цей рік. Для збору цих даних та парсингу JSON файлів була використана стороння бібліотека cve-manager на мові програмування Python. У таблиці 3.1. відображено кодування строкових полів у цілі числа.

Для того, щоб застосувати мітки до навчальних даних, довідкову карту CVE (описану у розділі 2), було завантажено (з використанням бібліотеки Requests) та проаналізовано (з використанням бібліотеки BeautifulSoup на мові Python). У цій довідковій таблиці подано визначення того, чи використовувалась у певних цілях та чи інша вразливість із бази даних NVD. Після того, як запис був взятий з файлу NVD, номер ідентифікатора CVE був вилучений та перевірений у відповідності до таблиці. Якщо було знайдено збіг (що вказує на існування експлуатації з використанням цієї вразливості), мітка «1» додається до списку ознак для поточного запису. В іншому випадку додається мітка «0», що вказує на відсутність наявної експлуатації. Потім цей список ознак (включаючи бінарну мітку) записується у CSV файл і формується фінальний датасет.

3.2 Процес побудови та тренування моделі

Наступний крок у реалізації включає побудову різних моделей нейронної мережі, які були протестовані. Бібліотеки, параметри та структура цих моделей описуються у даному підрозділі. Реалізація здійснювалася на мові програмування Python.

Для нейронної мережі глибокого навчання була обрана бібліотека Keras завдяки простоті у її використанні для побудови моделей машинного навчання. Keras - це бібліотека що працює на базі відомої бібліотеки Tensorflow. TensorFlow - це потужна бібліотека машинного навчання з відкритим кодом, що розроблена та опублікована Google. Keras дозволяє регулювати багато параметрів нижнього рівня для створення нейронної мережі та розбиває конструкцію на кілька простих кроків. Keras також містить

набір попередньо побудованих класів моделей, і для цієї роботи було обрано клас послідовних моделей, оскільки він моделює лінійний стек шарів. Даний стек - це доволі традиційне виконання нейронних мереж глибокого навчання. Після створення послідовної моделі, шари можуть додаватись окремо, для отримання фінальної ефективної реалізації.

Як було сказано у попередньому розділі, DNN, що використовується в цій роботі, має три основні шари: вхідний шар, один прихований шар та вихідний шар. Крім того, були додані «dropout»-шари, до та після прихованого шару. «Dropout»-шар - це спеціалізований шар, який використовується для запобігання перенавчання моделі, у якій він використовується, деактивуючи встановлений відсоток випадково вибраних вузлів протягом кожної навчальної епохи. Це змушує мережу використовувати різні вузли в кожну епоху, і зменшує ймовірність того, що мережа просто "запам'ятовує" навчальні дані та перенавчається [25].

Для тренування моделі, першим кроком є завантаження даних з файлу CSV, в якому вони зберігалися, що було виконано за допомогою бібліотеки обробки даних Pandas. Pandas також був використаний для розділення даних на набір ознак та відповідних міток, а бібліотека scikit-learn була використана для поділу даних на навчальну та тестову вибірки (80% навчальних даних, 20% для тестування). В датасеті були в наявності приблизно 60000 прикладів (починаючи з 2012 по 2018 рік). Приблизно 45 000 записів були використані для навчання, а решта 15 000 - для тестування. Навчальні дані подавались на вхід моделі, кожна з яких має функцію, яка приймає набір даних і повертає пристосовану модель. У випадку з DNN, потрібно також вказати кілька додаткових параметрів для функції тренування: кількість епох (ітерації тренувального і валідаційного процесу), розмір партії (кількість зразків, що розповсюджуються через мережу), і розділення валідації (відсоток навчальних даних, що використовується для тестування після кожної епохи). Ці параметри також підбираються методом проб та помилок, але в якості вихідного пункту було обрано набір з 100 епох, розмір партії 1024 та процент валідації - 20%.

Після того як модель пройшла тренування, вихідні дані були передані іншій функції для оцінки. З використанням бібліотек, потрібен один виклик функції (з даними тестування та мітками в якості параметрів), щоб визначити оцінку точності. Ці функції, характерні для моделі, були викликані в іншій функції, яка повторює обчислення оцінки кількість разів, визначену користувачем, і повертає середнє значення. Дані оцінки детально описані в наступному підрозділі.

3.3 Результати роботи нейронної мережі та порівняння результатів при різних конфігураціях

В даному розділі будуть наведені результати навчання мережі при використанні різних параметрів та конфігурацій.

При досліджені отриманого фінального датасету, було виявлено, що дані є дуже розбалансовані. Процент «експлуатованих» вразливостей від всіх наявних становив 6.52%. З метою того, щоб трохи «підвищити» даний процент - були відкинуті дані за 2019 та 2020 роки, оскільки більшість з нещодавно виявлених загроз з меншою ймовірністю використані деінде. Таким чином, процент використаних вразливостей склав 7.56%.

При тренуванні і тестуванні моделі, параметри якої описані у попередньому розділі, через явну перевагу «нульових» міток - вектор передбачення також складається з нульового вектора, оскільки, для обраного датасету така стратегія поведінки дає максимальну точність на тестовій вибірці (92,4%). На рисунку 3.2. відображена матриця помилок моделі.

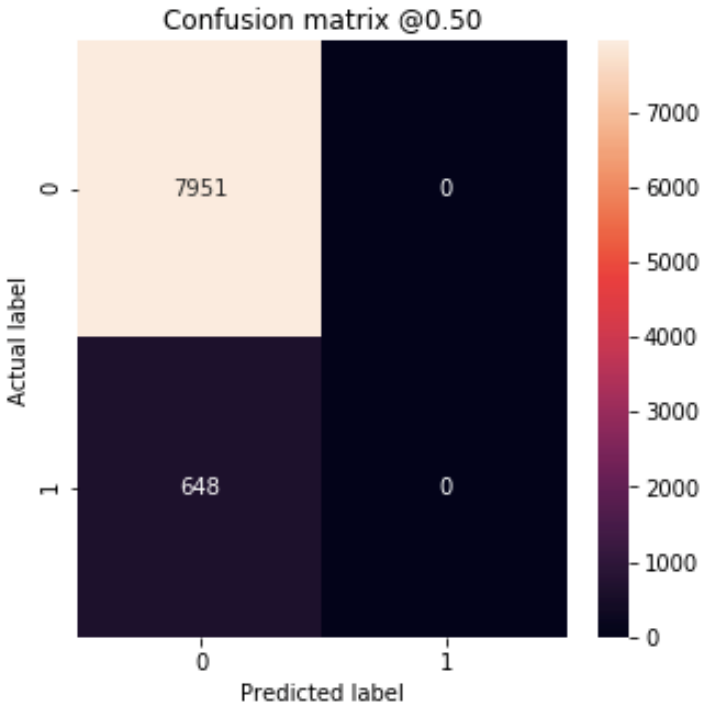

Рисунок 3.2 - Матриця помилок першої моделі

Не дивлячись на високий показник точності, для використання на практиці така модель є абсолютно неприйнятна, так як вона може пропустити майже всі потенційні загрози.

Процес тренування мережі при такій різниці в пояснюваній змінній доволі непростою задачею. Тому, до моделі додамо зміщення (3.1), яке залежить від відношення елементів классу «1» (true) до всіх елементів.

$$p_0 = pos/(pos + neg) = 1/(1 + e^{-b_0})$$

$$b_0 = -ln(1/p_0 - 1)$$

$$b_0 = ln\left(\frac{pos}{neg}\right) \quad (3.1)$$

де *pos* - кількість використаних вразливостей, *neg* - кількість невикористаних вразливостей. Отримуємо: $b_0 = -2.50$

Щоб прийняти рішення щодо використання зміщення, порівняємо графіки функціїї втрат для моделі з використанням зміщення, та для моделі без використання зміщення. Результат відображено на рис. 3.3. Параметри запуску:

- величина «пакета» = 1024;
- learning_rate = 0.001 (надалі не буде змінюватися);
- кількість епох = 20;
- значення «dropout» шарів = 0.2.

Таким чином, можна бачити, що з використанням зміщення - функція втрат збігається швидше, тому дане зміщення буде використовуватися і надалі.

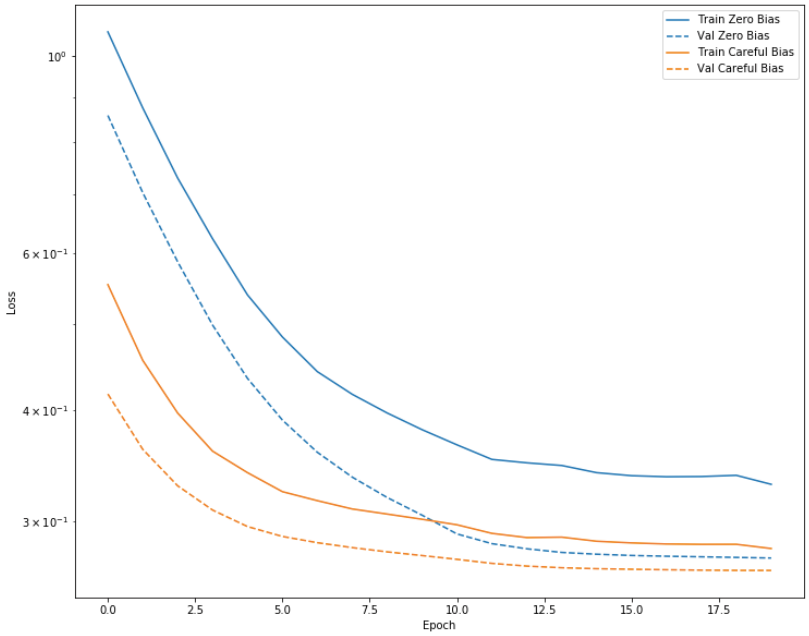

Рисунок 3.3 - Графік функцій втрат моделі з використанням та без використання зміщення

Тим не менш, додавання зміщення не вирішує проблеми нерівномірного розподілу датасету, тому залишається потреба у використанні допоміжного інструменту для балансування прогнозів мережі. В якості такого інструменту,

використаємо встановлення додаткових ваг на класи «0» та «1» - відповідей мережі.

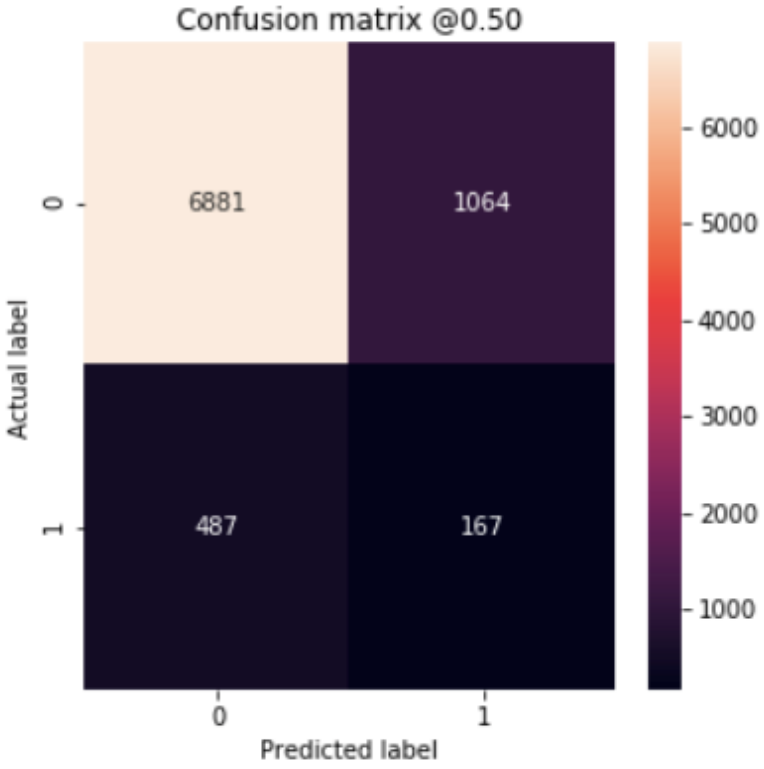

Рисунок 3.4 - Матриця помилок моделі №2

Задля того, щоб ваги корелювали з відношенням кількості елементів кожного класу до загальної кількості елементів у датасеті - скористаємося наступними формулами:

$$w_0 = \frac{total}{2*neg}$$

$$w_1 = \frac{total}{2*pos}$$

де pos - кількість елементів у класі «1», neg - кількість елементів у класі «0», а total - загальна кількість елементів у датасеті. Отримуємо: $w_0 = 0.54$, $w_1 = 6.61$.

У новій моделі застосуємо отримані корегуючі ваги, кількість епох збільшимо до 80, інші параметри залишимо аналогічними попереднім. Матриця помилок другої моделі відображена на рисунку 3.4. Можна бачити, що з даними налаштуваннями, модель показує кращий результат, та все ж відносно велика кількість «загроз» все ж залишається невиявленою.

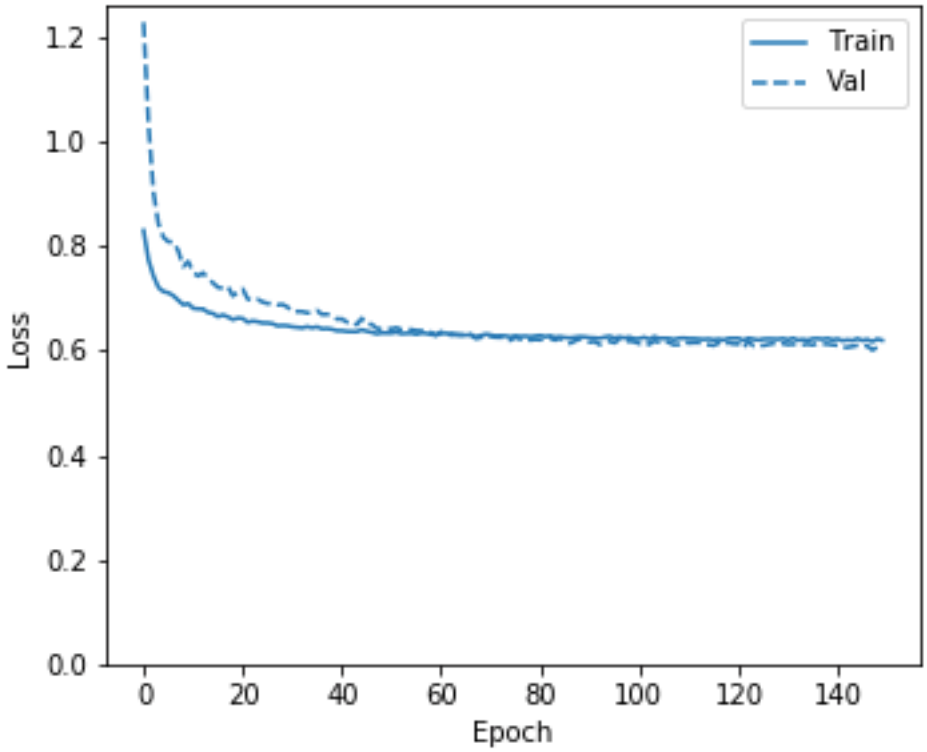

Рисунок. 3.5 - Функція втрат моделі №3

У новій моделі застосуємо отримані корегуючі ваги, кількість епох збільшимо до 150, а розмір «пакета» збільшимо до 2048. Також, потребують корегування ваги класів (зменшення ваг класу «1» на 1,5). Результати роботи моделі відображені на рис. 3.5. (графік функції втрат) та рис. 3.6. (матриця помилок). Схема самої нейронної мережі, що використовується у фінальній моделі, зображена на рис. 3.7.

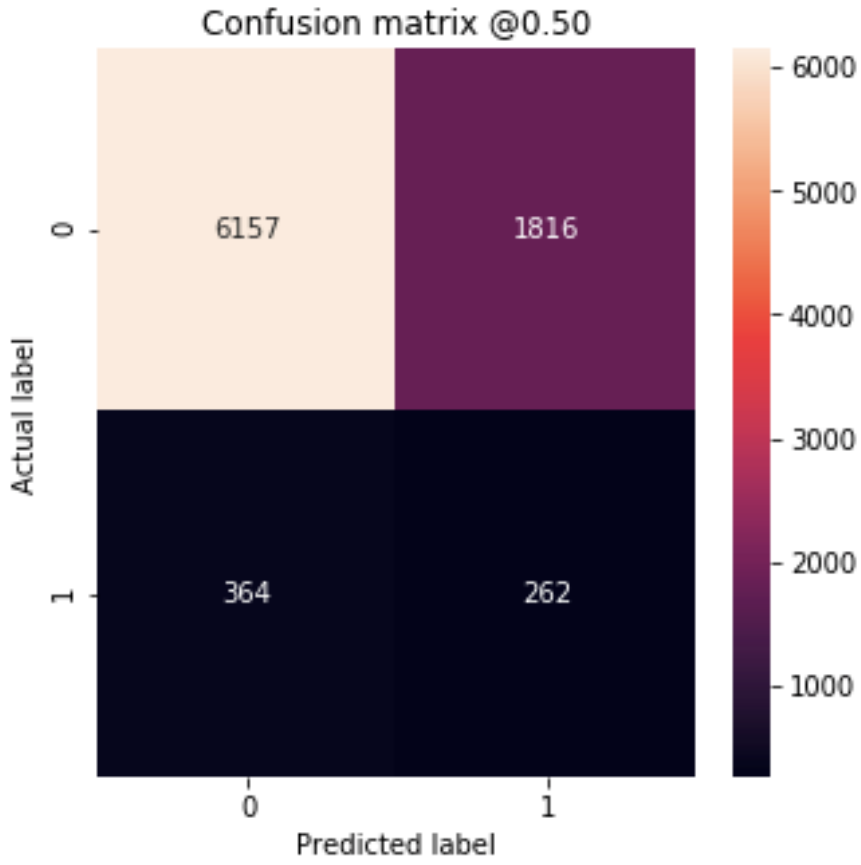

Рисунок 3.6 - Графік помилок третьої моделі

При подальших змінах вагів, зміщень та коригуваннях інших вхідних параметрів у пошуку конфігурації моделі, що підвищила б кількість True Positives відповідей моделі - починала суттєво збільшуватися кількість помилок першого роду, що негативно відображувалося на всіх метриках точності, тому дана конфігурація моделі визначена як оптимальна, при наявності датасету, у якому спостерігається такий дисбаланс.

Таблиця 3.1 - Порівняльна таблиця метрик точності побудованих моделей

	Accuracy	**Precision**	**Recall**
Модель №1	92.4%	0 %	0 %
Модель №2	81.9%	13.5%	25.5%
Модель №3	74.6%	12.6%	41.8%

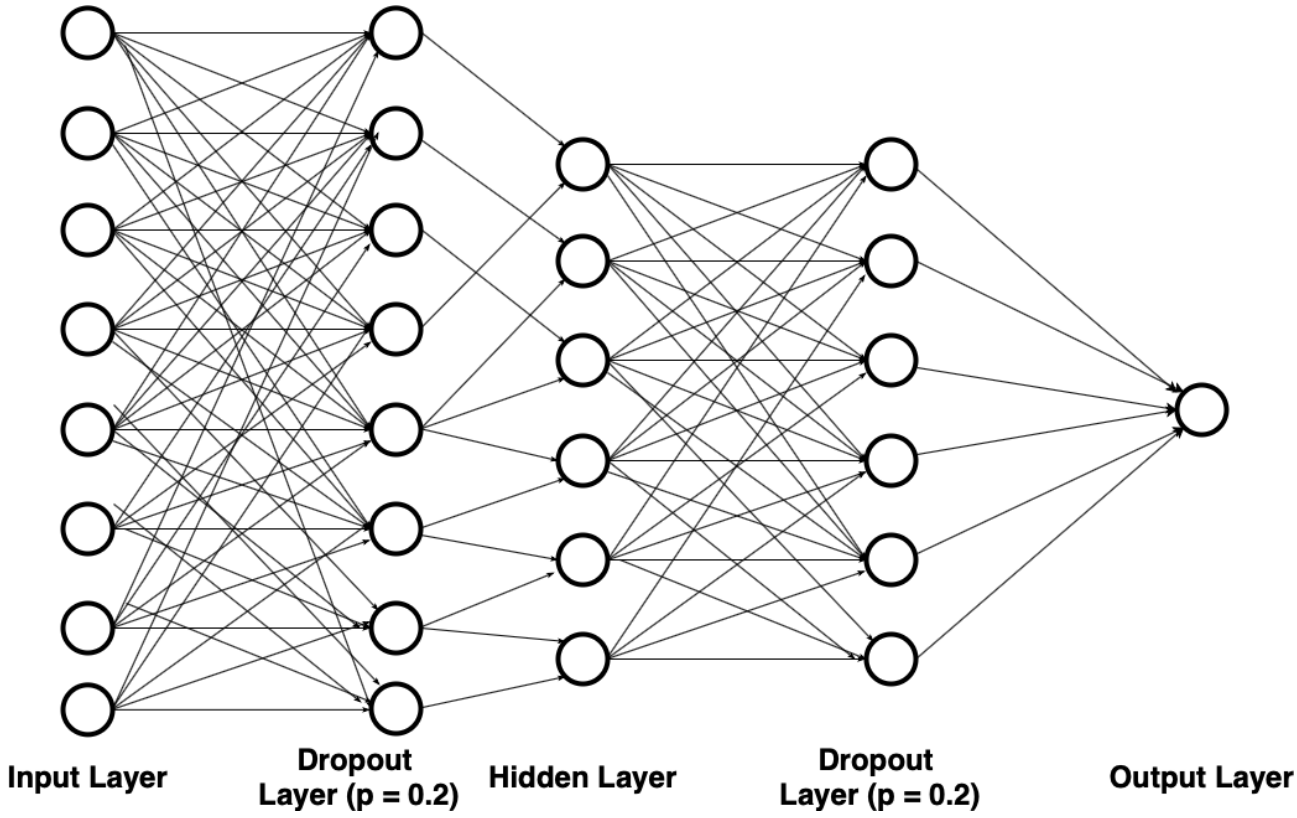

Рисунок 3.7 - Схема фінальної конфігурації нейронної мережі

3.4 Висновок до розділу 3

У даному розділі були описані експериментальні дослідження, що стосувалися побудови ШНН. Надані результати бінарної класифікації, які виконувала мережа згідно різних метрик точності, формули яких були наведені у розділі №2.

Також, був детально розглянутий процес обробки вхідних даних з баз даних: NVD (National Vulnerability Databese) та Exploit-DB.

Через дисбаланс, що можна спостерігати у результуючому датасеті (кількість записів, що відносяться до классу «True» становила 7.5% від загальної кількості елементів датасету), виникла потреба значних корегувань у архітектурі роботи моделі. Таким чином, було введене зміщення (bias) величиною -2.5 та додані додаткові ваги на класи відповідей {True, False}. Це

значно покращило результати роботи, та забезпечило доволі гарну точність (74%) за метрикою accuracy, що дозволяє використовувати побудовану мережу для практичних проблем у сфері інформаційної безпеки.

РОЗДІЛ 4 ЕКОНОМІЧНА ЧАСТИНА. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

4.1 Постановка завдання техніко-економічного дослідження

Розроблюваний програмний продукт є системою, що визначає за параметрами певної вразливості комп'ютерної мережі, чи може вона бути використана у процесі кібератаки чи у будь-якій спробі зашкодження цій мережі. Система дозволяє завантажувати дані про нові загрози, оновлювати дані з баз даних існуючих загроз шляхом парсингу сайтів та використання сторонніх бібліотек написаних на мові програмування Python.

Щоб залишатися конкурентоспроможною організацією у сфері захисту інформації, необхідно мати зручний і надійний інструмент, який зможе автоматизувати роботу у сфері діяльності організації. Задачею проекту було створити програму з оптимальним набором необхідних функцій і простим інтерфейсом, що не потребував би великої кількості часу на освоєння та обробку інформації.

4.2 Обґрунтування функцій та параметрів програмного продукту

Виходячи з конкретних цілей, які реалізуються :

F1 - вибір мови програмування для побудови ШНН (штучної нейронної мережі): а) Python, б) R;

F2 - вибір способу доступу, зберігання та використання даних: а) CSV-файли; б) База даних (БД) Postgresql;

F3 - парсинг даних: а) самостійно (в залежності від обраної мови програмування), б) з використанням готових бібліотек.

Виходячи з представлених варіантів будуємо морфологічну карту (рис.4.1).

Спираючись на карту - побудована позитивно-негативна матриця (табл. 4.1).

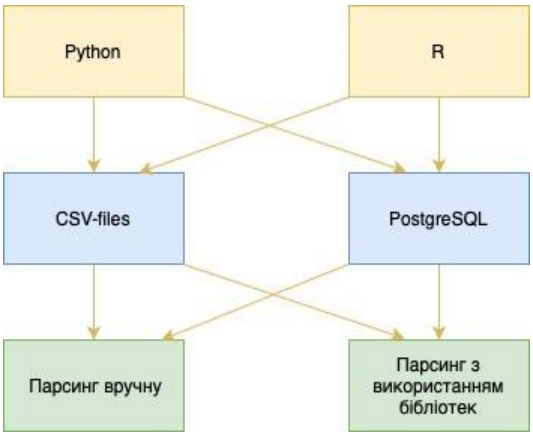

Рисунок 4.1 - Морфологічна карта.

Таблиця 4.1 - Позитивно-негативна матриця

Основна функція	**Варіант реалізації**	**Переваги**	**Недоліки**
F1	А	Об’єктно орієнтованість, бібліотеки для більш зручної роботи	Незручності в процесі відладки коду
	Б	Більші можливості візуалізацій та статистичних обрахунків	Використання більших об’ємів пам’яті при використанні
F2	А	Легкість забезпечення середи для зберігання	Незручності при об’єднанні та доступу до даних таблиць
	Б	Складність та потреба у багатьох сторонніх ПЗ для забезпечення середи зберігання	Швидкість та зручність доступу до даних та створення моделі, стабільність у використанні
F3	А	Витрати пам’яті, можливість підлаштувати під власну архітектуру	Потребує значних часових витрат для розуміння і роботи з вебом (HTML, CSS)

	Б	Гарантія отримання всіх потрібних даних	Складність інтегрування з іншою частиною власного ПЗ

Для характеристики прототипу програмного додатку використовуємо параметри X1 – X4. На основі даних, що представлені у літературі, визначаємо мінімальні, середні отримуванні та максимально допустимі значення (табл. 4.2)

Таблиця 4.2 - Система параметрів додатку

Назва параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	нс/Оп	360	100	12
Об’єм пам’яті для коректної роботи ПЗ	X2	Мб	64	32	8
Час тренування мережі	X3	мс	4000	1500	400
Потенційний об’єм програмного коду	X4	кількість рядків коду	1400	900	500

В даному випадку, для функції F1 застосовуються параметри X1. Для F2 – X2 і для F3, відповідно, - X3 та X4.

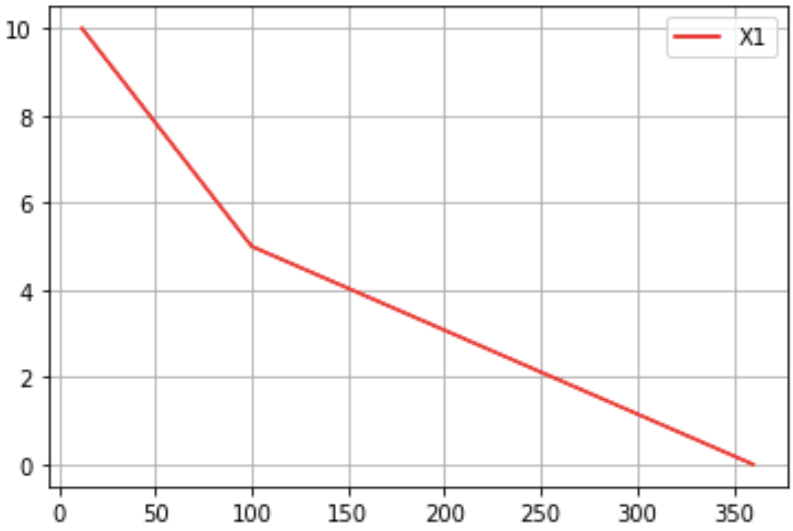

Рисунок 4.2 - Значення параметру X1

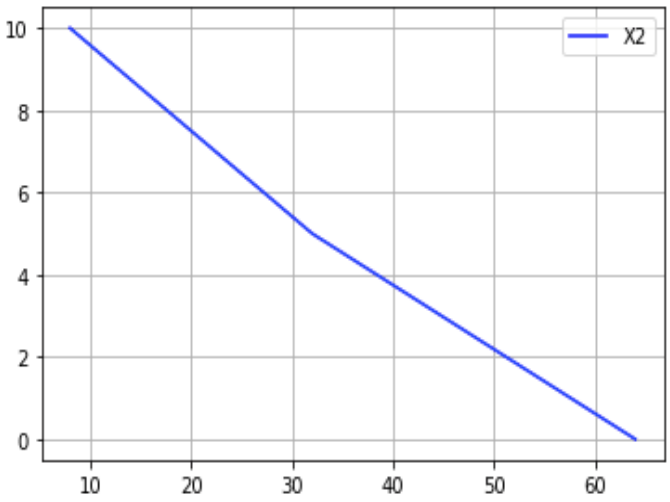

Рисунок 4.3 – Значення параметру X2

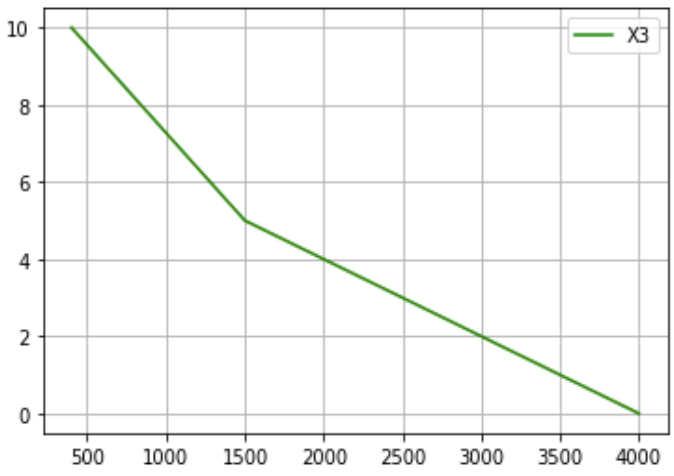

Рисунок 4.4 – Значення параметру X3

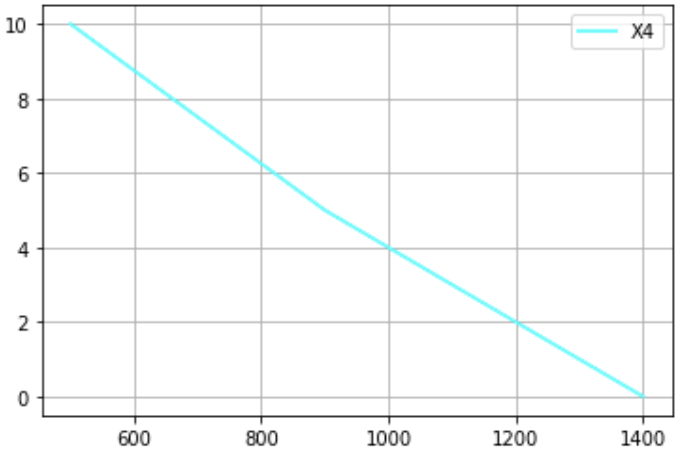

Рисунок 4.5 – Значення параметру X4

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Ранги варіюються від 1 до 5. Результати наведені в табл. 4.3-4.4.

Таблиця 4.3 - Результат оцінки параметрів

Познач. параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	нс/Оп	2	3	1	2	1	1	2	12	-5,5	30,25
X2	Об’єм пам’яті для коректної роботи	Мб	1	2	2	4	3	2	3	17	-0,5	0,25
X3	Час тренування мережі	Мс	3	1	3	1	2	3	1	14	-3,5	12,25
X4	Потенційний об’єм	к-сть рядків коду	4	4	4	3	4	4	4	27	9,5	90,25

	програмного коду											
	Разом		10	10	10	10	10	10	10	70	0	133

Таблиця 4.4 - Попарне зрівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	<	<	<	<	0,5
X1 і X3	<	>	<	>	<	<	>	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	>	<	>	>	<	>	>	1,5
X2 і X4	<	<	<	>	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Порахуємо коефіцієнт конкординації наступною формулою.

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 133}{7^2(4^3 - 4)} = 0{,}88 > W_k = 0{,}67$$

Так як коефіцієнт конкординації більше нормативного, результати вважають достовірними.

Розрахунок вагомості параметрів наведено в табл. 4.5.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	1,5	1,5	5,5	0,319	20,25	0,116	100	0,215
X2	0,5	1,0	0,5	1,5	3,5	0,181	24,25	0,382	124,25	0,283
X3	0,5	1,5	1,0	1,5	4,5	0,444	37,25	0,247	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,056	18,25	0,255	64,75	0,154
Всього:					16	1	100	1	445	1

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Визначаємо рівень якості кожного варіанту виконання окремо.

Абсолютні значення параметрів *X2* (об'єм необхідної оперативної пам'яті) та *X3* (час тренування нейромережі) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1 (*швидкодія мови програмування*)* обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 100 нс/Оп або варіанту б) 12 нс/Оп.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	A(X1)	170	3,6	0,215	0,874
F2(X2)	A(X2)	45	3,4	0,283	0,764
F3(X3,X4)	А (X3)	2700	2,4	0,348	0,685
	А (X4)	1000	4	0.155	0,62
	Б (X3)	3050	1.8	0.348	0,63
	Б (X4)	1400	1	0,155	0,288

За даними з таблиці 4.6 визначаємо рівень якості кожного з варіантів за наступними формулами.

$$K_{я1} = 0.874 + 0.764 + 0.685 + 0.62 = 2.943$$
$$K_{я2} = 0.874 + 0.764 + 0.63 + 0.288 = 2.556$$

Оскільки варіант 1 має найбільший коефіцієнт якості, він є найкращим.

5.3 Економічний аналіз варіантів розробки

Для оцінки трудомісткості розробки спочатку проведемо розрахунок трудомісткості. Усі варіанти мають наступні основні завдання:

1. Розробка проекту програмного продукту:

А) для першого варіанту реалізації використовуємо Python;

Б) на мові програмування R.

2. Розробка програмної оболонки;

Завдання 1.А) має наступні показники: ступінь новизни В, група складності 1, для реалізації використана довідкова інформація. Таким чином: $Т_Р$ = 43, $К_П$ = 0.81, $К_{СК}$ = 1,$К_{СТ}$ = 0.6, $К_{СТ.М}$ = 1. $T_{1А} = 43 * 0.81 * 1 * 0.6 * 1 = 21$ людино-день.

Завдання 1.Б) за ступенем новизни відноситься до групи А, групи складності – 1, для реалізації використана довідкова інформація. Таким чином: $Т_Р$ = 90, $К_П$ = 1.70, $К_{СК}$ = 1, $К_{СТ}$ = 0.8, $К_{СТ.М}$ = 1. $T_{1Б} = 90 * 1.7 * 1 * 0.8 * 1 = 122.4$ людино-дня.

Для другого завдання (алгоритм третьої групи складності, степінь новизни Б, використовує інформацію у вигляді даних) маємо наступні параметри: $Т_Р$ = 28, $К_П$ = 0.7,$К_{СК}$ = 1,$К_{СТ}$ =0,8, $К_{СТ.М}$ = 1. $T_2 = 28 * 0.7 * 1 * 0.8 * 1 = 15.68$ людино-дня.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість (у людино-годинах) та відображуємо розрахунки у наступних формулах.

$$T_I = (21 + 15.68) * 8 = 293.44$$

$$T_{II} = (122.4 + 15.68) * 8 = 1104.64$$

Найбільш високу трудомісткість має другий варіант.

В розробці беруть участь аналітик та програміст з окладом 6000 та 7000 грн відповідно. Визначимо зарплату за годину за такою формулою.

$$С_ч = \frac{М}{T_m \cdot t} грн.$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$СЧ = \frac{13000}{2*21*8} = 38,69$$

Тоді, розрахуємо заробітну плату за формулою - $С_{ЗП} = С_ч \cdot Т_i \cdot К_Д$, де $С_ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників становить:

I. $С_{ЗП} = 38.69 \cdot 293.44 \cdot 1.2 = 13623$ грн.

II. $С_{ЗП} = 38.69 \cdot 1104.64 \cdot 1.2 = 51166$ грн.

Відрахування на соціальний внесок становить 22,0%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 13623 \cdot 0,22 = 2997,06$

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 51166 \cdot 0,22 = 11256,52$

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 7000 \cdot 0,2 = 16\,800 грн.$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 16800 \cdot (1 + 0.2) = 20160 грн.$$

Відрахування на соціальний внесок:

$$С_{ВІД}= С_{ЗП} \cdot 0.22 = 20160* 0,22 =4435,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн. Тоді отримаємо:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 25000 = 7187,5 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 25000 \cdot 0.05 = 1437,5 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706,4 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,8 \cdot 0,2 \cdot 1,75= 477,792 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0,67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть визначатися наступним чином.

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н}$$

$C_{ЕКС}$ = 20160 + 4435,2+ 7187,5 + 1437,5 + 477,792 + 16750 = 50557,992 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме $C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ}$ = 50557,992 / 1706,4 = 29,62 грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$C_{М} = C_{М\text{-}Г} \cdot T$$

I. $C_{М} = 28{,}69 \cdot 293{,}44 = 8418{,}79$ грн.

II. $C_{М} = 28{,}69 \cdot 1104{,}64 = 31692{,}12$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0{,}67$$

I. $C_{Н} = 8418{,}79 \cdot 0{,}67 = 5640{,}58$ грн

II. $C_{Н} = 31692{,}12 \cdot 0{,}67 = 21233{,}72$ грн.

Отже, вартість розробки ПП становить $C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}$:

I. $C_{ПП} = 13623 + 2997{,}06 + 8418{,}79 + 5640{,}58 = 30679{,}43$ грн.

II. $C_{ПП} = 51166 + 11256{,}52 + 31692{,}12 + 21233{,}72 = 115348{,}36$ грн.

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj}$$

$$К_{ТЕР1} = 2.943 / 30679,43 = 9,59 \cdot 10^{-5};$$

$$К_{ТЕР2} = 2.556 / 115348,36 = 2,26 \cdot 10^{-5};$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 9,59 \cdot 10^{-5}$.

4.4 Висновок до розділу 4

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного

комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 9{,}59 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- спосіб доступу, зберігання та використання даних – CSV-файли;
- парсинг даних – самостійно.

ВИСНОВОК

Метою даної дипломної роботи було вивчення методів створення інструменту, заснованого на навчанні, для прогнозування експлуатації вразливостей на основі їх особливостей. Для перевірки їхньої ефективності була реалізована нейронна мережа глибокого навчання (DNN). Для того, щоб створити набір навчальних даних для моделей, інформація була отримана як з National Vulnerability Database та з Offensive Security Exploit Database. Після того, як ця інформація була зібрана та відформатована, було вибрано функції для надання найбільш точної інформації для оцінки моделей. Обрані функції включали інформацію, таку як вектор доступу вразливості, складність доступу, бал CVSS та інші. У поєднанні з цими функціями використовувалася довідкова таблиця CVE для узгодження відомих вразливих місць із відомими кібератаками, яка забезпечувала цільові мітки для навчальних даних (двійкові). Ці функції та мітки потім використовувались для навчання та тестування різних моделі.

Оскільки датасет був дуже несбалансований, тобто кількість існуючих значно перевищує кількість вразливостей що деінде використовувались - найвища точність, яку вдалося досягти - 80%, але з врахуванням всіх метрик точності, с поставленою задачею краще впорався варіант моделі з показником точності 74%. Це говорить про те, що ця модель може бути розроблена в інструменті, який може надавати допомогу системним адміністраторам та іншим фахівцям із безпеки, та все ж вона потребує певних доробок.

Щодо вдосконалень, та перспективних рішень, які можуть бути застосовані до цієї роботи - це пошук нових джерел інформації саме про використані вразливості, оскільки, як відомо, якість та повнота датасету є одним із найважливіших параметрів побудови даної моделі. Також, дана модель потребує розробки власної функції втрат, у якій штраф за False Positive відповідь буде вище і таким чином, це додатково встановить баланс у прогнозі нейронної мережі. Також, дана модель може бути інтегрована у програмному

продукті зі сканерами безпеки, що будуть подавати на вхід інформацію про вразливості та отримувати відповідь щодо того, чи може дана вразливість дійсно бути використана кіберзлочинцями.

СПИСОК ЛІТЕРАТУРИ

1. Гатчин Ю.А.,Сухостат В.В., Куракин А.С., Донецкая Ю.В. Теория информационной безопасности и методология защиты информации. Санкт-Петербург: Университет ИТМО, 2018. 100 с.

2. Mouna Jouini, Latifa Ben Arfa Rabai, Anis Ben Aissa Classification of Security Threats in Information Systems. *Procedia Computer Science,* 2014. №32, с. 289-296

3. Ehrenfeld, Jesse M. WannaCry, Cybersecurity and Health Information Technology: A Time to Act, *Journal of Medical Systems 41,* 2017, №7. URL doi:http://0- dx.doi.org.library.uark.edu/10.1007/s10916-017-0752-1 (дата звернення: 10.05.2020).

4. Microsoft. Microsoft Security Bulletin MS17-010: Critical. URL: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010 (дата звернення: 10.05.2020).

5.Марков А.С., Фадин А.А. Организационно-технические проблемы защиты от целевых вредоносных программ типа Stuxnet. *Вопросы кибербезопасности,* 2013. № 1(1). С. 28-36.

6. Swiderski F, Snyder W. Threat Modeling. Microsoft Press. 2004. 288 p.

7. Andreea Bendovschi. «Cyber-Attacks – Trends, Patterns and Security Countermeasures», *Procedia Economics and Finance,* 2015. №7, p. 12-15.

8. SQL-ін’єкція. URL: https://uk.wikipedia.org/wiki/SQL-ін%27єкція (дата звернення: 17.05.2020)

9. 1. Криптографические методы защиты информации – URL: https://sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema1 (дата звернення: 17.05.2020)

10. PCI Security Standards Council. URL: https://www.pcisecuritystandards.org/ (дата звернення: 17.05.2020)

11. Common Vulnerability Scoring System URL: http://www.first.org/cvss/ (дата звернення: 17.05.2020)

12. Хабрахабр. *Сканирование на уязвимости и безопасная разработка. Часть 1*. URL: https://habr.com/ru/post/444534/ (дата звернення: 17.05.2020)

13. Лаборатория Качества. *Особенности тестирования «серого ящика»*. URL: https://quality-lab.ru/blog/key-principles-of-gray-box-testing/ (дата звернення: 17.05.2020)

14. Журнал «Компьютеры, Сети, Программирование» 01/2009, с.19 URL: https://books.google.com.ua/books?id=GZG2edo-ECwC&source=gbs_book_other_versions_r&redir_esc=y (дата звернення: 17.05.2020)

15. Люггер Джордж Ф. Искусственный интеллект: стратегии и методы решения сложных проблем. / Москва: Издательский дом «Вильямс», 2004. – 864 с.

16. National Vulnerability Database. *"Common Vulnerabilities and Exposures" Information Technology Laboratory, National Institute of Standards and Technology*. URL: https://cve.mitre.org/index.html (дата звернення: 17.05.2020)

17. Deng, L.; Yu, D. (2014). "Deep Learning: Methods and Applications". *Foundations and Trends in Signal Processing*, 2014. №7. p. 20–28.

18. Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2009. №2, p. 80-127.

19. Hinton, G. E. A Practical Guide to Training Restricted Boltzmann Machines. Toronto: Department of Computer Science, University of Toronto, 2010. 20 p.

20. Recurrent neural network based language model / Mikolov, T et. al., Japan: INTERSPEECH-2010, 2010. 12 p.

21. Goh, A.T.C. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 2009 №9. p. 143-151.

22. Hawking, D.M. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 2012. №44 (1). p. 1-12.

23. Cao, W. Mirchandani, G. On hidden nodes for neural nets. *IEEE Transactions on Circuits and Systems*, 1989. №36 (5). p. 661-664.

24. Keras Documentation. *Metrics.* URL: https://keras.io/metrics/ (дата звернення: 17.05.2020)

25. Geoffrey E. Hinton, Srivastava Nitish, Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors, 2012 arXiv:1207.0580 URL: https://arxiv.org/abs/1207.0580 (дата звернення: 17.05.2020)

26. Xavier Glorot, Antoine Bordes, Yoshua Bengio. Deep sparse rectifier neural networks. Montreal, University of Montreal: *AISTATS*, 2011. №11. p. 315-323.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

parsing_cve.py

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
vgm_url = 'https://cve.mitre.org/data/refs/refmap/source-EXPLOIT-DB.html'
html_text = requests.get(vgm_url).text
soup = BeautifulSoup(html_text, ‘html.parser')
my_table = soup.find('div', {'id' : «CenterPane"})
links = my_table.findAll(‘tr')
cve_exploits = []
for link in links:
    for l in link.findAll('a'):
        cve_exploits.append(l.string)
exploits = pd.DataFrame(cve_exploits[1:])
exploits = exploits.rename(columns={0: «cve_id"})
exploits.to_csv(‘my_exploits.csv')
```

parsing_nvd.py

```
import json
import pandas as pd
import numpy as np
cve2020 = pd.read_json('nvd_bd/nvdcve-1.1-2010.json')
cve2019 = pd.read_json('nvd_bd/nvdcve-1.1-2011.json')
cve2018 = pd.read_json('nvd_bd/nvdcve-1.1-2018.json')
cve2017 = pd.read_json('nvd_bd/nvdcve-1.1-2017.json')
cve2016 = pd.read_json('nvd_bd/nvdcve-1.1-2016.json')
cve2015 = pd.read_json('nvd_bd/nvdcve-1.1-2015.json')
cve2014 = pd.read_json('nvd_bd/nvdcve-1.1-2014.json')
cve2013 = pd.read_json('nvd_bd/nvdcve-1.1-2013.json')
```

```
        cve2012 = pd.read_json(‘nvd_bd/nvdcve-1.1-2012.json')
        full_cve = pd.concat([cve2018,cve2017,cve2016,cve2015,cve2014,cve2013,
cve2012, cve2011, cve2010], ignore_index = True)
        cve20 = full_cve[['CVE_Items']]
        cve20['cve_id'] = np.nan
        cve20['cve_id'] = cve20['cve_id'].astype(str)
        cve20['complexity'] = np.nan
        cve20['complexity'] = cve20['complexity'].astype(str)
        cve20['vector'] = np.nan
        cve20['vector'] = cve20['vector'].astype(str)
        cve20['integrity'] = np.nan
        cve20['integrity'] = cve20['integrity'].astype(str)
        cve20['available'] = np.nan
        cve20['available'] = cve20['available'].astype(str)
        cve20['confident'] = np.nan
        cve20['confident'] = cve20['confident'].astype(str)
        cve20['cvss'] = np.nan
        cve20['version'] = np.nan
        cve20['exploitability'] = np.nan
        cve20['impact'] = np.nan
        for index, row in cve20.iterrows():
            cve20.at[index,'cve_id']                                    =
row['CVE_Items']['cve']['CVE_data_meta']['ID']
            if 'baseMetricV3' in row['CVE_Items']['impact'].keys():
                cve20.at[index,'vector']                                =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['attackVector']
                cve20.at[index,'complexity']                            =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['attackComplexity']
                cve20.at[index,'confident']                             =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['confidentialityImpact']
```

```
            cve20.at[index,'integrity'] =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['integrityImpact']
            cve20.at[index,'available'] =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['availabilityImpact']
            cve20.at[index,'cvss'] =
row['CVE_Items']['impact']['baseMetricV3']['cvssV3']['baseScore']
            cve20.at[index,'exploitability'] =
row['CVE_Items']['impact']['baseMetricV3']['exploitabilityScore']
            cve20.at[index,'impact'] =
row[‘CVE_Items’]['impact']['baseMetricV3']['impactScore']
        else:
            continue
        nvd_db = cve20[['cve_id', 'complexity', 'vector', 'integrity', 'available',
'confident', 'cvss', 'exploitability', ‘impact']]
        nvd_db = nvd_db.dropna()
        nvd_db.to_csv(‘nvd_db_v3_2012_2018.csv')
```

dnn.py

```
        import tensorflow as tf
        from tensorflow import keras
        import os
        import tempfile

        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import seaborn as sns

        import sklearn
```

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import backend as K
def plot_loss(history, label, n):
  # Use a log scale to show the wide range of values.
  plt.semilogy(history.epoch,  history.history['loss'],
         color=colors[n], label='Train '+label)
  plt.semilogy(history.epoch,  history.history['val_loss'],
      color=colors[n], label='Val '+label,
      linestyle="--")
  plt.xlabel('Epoch')
  plt.ylabel('Loss')

  plt.legend()

def plot_metrics(history):
  metrics =  ['loss', 'auc', 'precision', 'recall']
  for n, metric in enumerate(metrics):
    name = metric.replace("_"," ").capitalize()
    plt.subplot(2,2,n+1)
    plt.plot(history.epoch,  history.history[metric],  color=colors[0],
label='Train')
    plt.plot(history.epoch, history.history['val_'+metric],
         color=colors[0], linestyle="--", label='Val')
    plt.xlabel('Epoch')
    plt.ylabel(name)
    if metric == 'loss':
      plt.ylim([0, plt.ylim()[1]])
    elif metric == 'auc':
```

```
        plt.ylim([0.5,1])
      else:
        plt.ylim([0,1])

      plt.legend()
    def plot_cm(labels, predictions, p=0.5):
      cm = confusion_matrix(labels, predictions > p)
      plt.figure(figsize=(5,5))
      sns.heatmap(cm, annot=True, fmt="d")
      plt.title('Confusion matrix @{:.2f}'.format(p))
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
      print('Legitimate Transactions Detected (True Negatives): ', cm[0][0])
      print('Legitimate Transactions Incorrectly Detected (False Positives): ', cm[0][1])
      print('Fraudulent Transactions Missed (False Negatives): ', cm[1][0])
      print('Fraudulent Transactions Detected (True Positives): ', cm[1][1])
      print('Total Fraudulent Transactions: ', np.sum(cm[1]))
    mpl.rcParams['figure.figsize'] = (12, 10)
    colors = plt.rcParams[‘axes.prop_cycle'].by_key()['color']

    raw_df = pd.read_csv('nvd_db_v3_2012_2018.csv')
    cve_id = 'cve_id'
    raw_df = raw_df.drop(['Unnamed: 0'], axis=1)
    raw_df['exploited'] = np.nan

    usedInExploit2 = pd.read_csv('exploited_cve/my_exploits.csv')

    df = raw_df.copy()
```

```
    vector_dict  = {'LOCAL' : 3, 'NETWORK' : 1, 'ADJACENT_NETWORK': 2,
'PHYSICAL':4}
    complex_dict = {'LOW':1, 'HIGH':2}
    confIntegAvail_dict = {'HIGH':3, 'NONE':1, 'LOW':2}

    df['vector'] = df['vector'].apply(lambda x: vector_dict[x])
    df['complexity'] = df['complexity'].apply(lambda x: complex_dict[x])
    df['confident'] = df['confident'].apply(lambda x: confIntegAvail_dict[x])
    df['available'] = df['available'].apply(lambda x: confIntegAvail_dict[x])
    df['integrity'] = df['integrity'].apply(lambda x: confIntegAvail_dict[x])

    usedExpArray2 = np.array(usedInExploit2['cve_id'].unique())
    allCVE = np.array(df['cve_id'].unique())
    exploited = list(set(allCVE) & set(usedExpArray2))
    for index, row in df.iterrows():
      if df.loc[index,'cve_id'] in exploited:
        df.at[index,'exploited'] = 1
      else:
        df.at[index,'exploited'] = 0
    neg, pos = np.bincount(df['exploited'])
    total = neg + pos
    print('Examples:\n    Total: {}\n    Positive: {} ({:.2f}% of total)\n'.format(
      total, pos, 100 * pos / total))
    cleaned_df = df.copy()
    cleaned_df.pop(‘cve_id')
    train_df, test_df = train_test_split(cleaned_df, test_size=0.2)
    train_df, val_df = train_test_split(train_df, test_size=0.2)
    train_labels = np.array(train_df.pop('exploited'))
    bool_train_labels = train_labels != 0
    val_labels = np.array(val_df.pop('exploited'))
```

```
test_labels = np.array(test_df.pop('exploited'))
train_features = np.array(train_df)
val_features = np.array(val_df)
test_features = np.array(test_df)
scaler = StandardScaler()
train_features_cvss = scaler.fit_transform(train_features[:,5:])
val_features_cvss = scaler.transform(val_features[:,5:])
test_features_cvss = scaler.transform(test_features[:,5:])
train_features[:,5:] = train_features_cvss
val_features[:,5:] = val_features_cvss
test_features[:,5:] = test_features_cvss
print('Training labels shape:', train_labels.shape)
print('Validation labels shape:', val_labels.shape)
print('Test labels shape:', test_labels.shape)

print('Training features shape:', train_features.shape)
print('Validation features shape:', val_features.shape)
print('Test features shape:', test_features.shape)

METRICS = [
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.BinaryAccuracy(name='accuracy'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
    keras.metrics.AUC(name='auc'),
]
```

```
def make_model(metrics = METRICS, output_bias=None):
  if output_bias is not None:
    output_bias = tf.keras.initializers.Constant(output_bias)
  model = keras.Sequential([
      keras.layers.Dense(8, activation='relu',
          input_shape=(train_features.shape[-1],)),
          keras.layers.Dropout(0.2),
          keras.layers.Dense(6, activation='relu'),
          keras.layers.Dropout(0.2),
          keras.layers.Dense(1, activation='sigmoid',
                      bias_initializer=output_bias),  ])
  model.compile(
      optimizer=keras.optimizers.Adam(lr=1e-3),
      loss=keras.losses.BinaryCrossentropy(),
      metrics=metrics)
  return model
EPOCHS = 150
BATCH_SIZE = 2048
initial_weights = os.path.join(tempfile.mkdtemp(),'initial_weights')
model.save_weights(initial_weights)
initial_bias = np.log([pos/neg])
weight_for_0 = (1 / neg)*(total)/2.0
weight_for_1 = (1 / pos)*(total)/2.0
class_weight = {0: weight_for_0, 1: weight_for_1 - 1.5}
print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1 - 1.5))
weighted_model = make_model(output_bias = initial_bias)
weighted_model.load_weights(initial_weights)
weighted_history = weighted_model.fit(
    train_features,
```

```
            train_labels,
            batch_size=BATCH_SIZE,
            epochs=EPOCHS,
            validation_data=(val_features, val_labels),
            class_weight=class_weight)

        plot_metrics(weighted_history)
        train_predictions_weighted = weighted_model.predict(train_features,
batch_size=BATCH_SIZE)
        test_predictions_weighted = weighted_model.predict(test_features,
batch_size=BATCH_SIZE)
        weighted_results = weighted_model.evaluate(test_features, test_labels,
                                                   batch_size=BATCH_SIZE, verbose=0)
        for name, value in zip(weighted_model.metrics_names, weighted_results):
          print(name, ': ', value)
        print()
        plot_cm(test_labels, test_predictions_weighted)
```